

D10

EDMRST_Finish	10	(RStIntFin.c)
EDMRST_initialize.....	3	(RStIntFin.c)
EDMRST_ping	9	(RStIntFin.c)

```

2  /*****
3  **
4  ** File Name:  RSTntrlfin.c
5  **
6  ** Copyright (c) 1996, 1999 by EMC Corporation.
7  **
8  ** Purpose:
9  **      This module contains the Restore API functions to
10 **      initialize and terminate the restore operation.
11 **
12 ** Table of Contents:
13 **      -----
14 **      API Functions:
15 **          ERMNST_Initialize
16 **          ERMNST_Finish
17 **
18 **      Internal Functions:
19 **
20 **
21 ** Compile-Time Options:
22 **      This section must list any compile time definitions
23 **      which will affect this header.
24 **
25 **      *****/
26
27 /* The following provides an RCS id in the binary that can be located
28 ** with the what(1) utility.  The intent is to keep this short.
29 **
30 */
31 #ifndef lint
32 static char RCS_id[] = "RSTntrlfin.c"
33 " $Revision$ "
34 " $State$";
35
36 #endif
37
38 /*
39 ** Feature test switches.
40 ** Standard defines required to turn on OS features go here.
41 **
42 ** The following is required for code that uses POSIX APIs.
43 ** Remove for non-POSIX, non-portable code.
44 **
45 */
46 #define _POSIX_SOURCE 1
47
48 /*
49 ** System headers.
50 **
51 */
52 #include <pwd.h>
53
54
55 /*
56 ** Epoch headers.
57 **
58 */
59 #include <eb/eb_port.h>
60 #include <eb/rb_log.h>
61
62 /*
63 ** Local headers
64 **
65 */
66 #include <RSTntrl.h>

```

```

65 #include <RSTntrl_cm.h>
66
67 /*
68 ** Comm headers.
69 **
70 ** #include <restore/csc_ERMNSTntrl.h>
71 ** #include <restore/csc_ERMNSTntrlcm.h>
72 ** #include <restore/dispatch_demon.h>
73 ** #include <restore/restore_engine.h>
74 ** #include <edmlink/edmlink_api.h>
75
76
77 /* #defines, structures, typedefs local to this source file
78 **
79 */
80
81 /*
82 ** Global declarations
83 **
84
85 internalHandler handler = NULL;

```

Wed Oct 08 16:48:45 2008	EDMST_Initiate	Page 3 of 12
87	/*	
88	***** EDMST_Initiate: *****	
89	* This function takes care of all the initialization for a recovery	
90	* session. This macro will be called prior to any of the other functions	
91	* in the Recover API.	
92	* In the Recover API.	
93	* Parameters:	
94	* hostname {i} - The machine name of the server to use.	
95	* strid {i} - A handle to receive a pointer to this user's client	
96	* client {i} - handle for the Restore Engine connection.	
97	* If - The maximum number of seconds to wait for the connection	
98	* to the Restore Engine process to be completed.	
99	* *****	
100	*****	
101	*****	
102	*****	
103	*****	
104	errno_t	
105	hostname_t hostname; /* hostname */	
106	serverhandle_t *strid; /* strid */	
107	unsigned long timeout; /* timeout */	
108	{	
109	errno_t api_status = E_SUCCESS;	
110	/*	
111	* uid_t human_wid;	
112	* struct passwd *pw;	
113	* char *human_username;	
114	* re_init_args;	
115	* re_init_spec;	
116	* re_init_spec;	
117	* re_init_spec;	
118	* re_init_spec;	
119	* re_init_spec;	
120	* re_init_spec;	
121	* re_init_spec;	
122	* re_init_spec;	
123	* re_init_spec;	
124	* re_init_spec;	
125	* re_init_spec;	
126	* re_init_spec;	
127	* re_init_spec;	
128	* re_init_spec;	
129	* re_init_spec;	
130	* re_init_spec;	
131	* re_init_spec;	
132	* re_init_spec;	
133	* re_init_spec;	
134	* re_init_spec;	
135	* re_init_spec;	
136	* re_init_spec;	
137	* re_init_spec;	
138	* re_init_spec;	
139	* re_init_spec;	
140	* re_init_spec;	
141	* re_init_spec;	
142	* re_init_spec;	
143	* re_init_spec;	
144	* re_init_spec;	
145	* re_init_spec;	
146	* re_init_spec;	
147	* re_init_spec;	
148	* re_init_spec;	
149	* re_init_spec;	
150	* re_init_spec;	
151	* re_init_spec;	
152	* re_init_spec;	
153	* re_init_spec;	
154	* re_init_spec;	
155	* re_init_spec;	
156	* re_init_spec;	
157	* re_init_spec;	
158	* re_init_spec;	
159	* re_init_spec;	
160	* re_init_spec;	
161	* re_init_spec;	
162	* re_init_spec;	
163	* re_init_spec;	
164	* re_init_spec;	
165	* re_init_spec;	
166	* re_init_spec;	
167	* re_init_spec;	
168	* re_init_spec;	
169	* re_init_spec;	
170	* re_init_spec;	
171	* re_init_spec;	
172	* re_init_spec;	
173	* re_init_spec;	
174	* re_init_spec;	
175	* re_init_spec;	
176	* re_init_spec;	
177	* re_init_spec;	
178	* re_init_spec;	
179	* re_init_spec;	
180	* re_init_spec;	
181	* re_init_spec;	
182	* re_init_spec;	
183	* re_init_spec;	
184	* re_init_spec;	
185	* re_init_spec;	
186	* re_init_spec;	
187	* re_init_spec;	
188	* re_init_spec;	
189	* re_init_spec;	
190	* re_init_spec;	
191	* re_init_spec;	
192	* re_init_spec;	
193	* re_init_spec;	
194	* re_init_spec;	
195	* re_init_spec;	
196	* re_init_spec;	
197	* re_init_spec;	
198	* re_init_spec;	
199	* re_init_spec;	
200	* re_init_spec;	
201	* re_init_spec;	
202	* re_init_spec;	
203	* re_init_spec;	
204	* re_init_spec;	
205	* re_init_spec;	
206	* re_init_spec;	
207	* re_init_spec;	
208	* re_init_spec;	
209	* re_init_spec;	
210	* re_init_spec;	
211	* re_init_spec;	
212	* re_init_spec;	
213	* re_init_spec;	

Wed Oct 08 16:48:45 2008	EDMST_Initiate	Page 4 of 12
214	* re_init_spec;	
215	* re_init_spec;	
216	* re_init_spec;	
217	* re_init_spec;	
218	* re_init_spec;	
219	* re_init_spec;	
220	* re_init_spec;	
221	* re_init_spec;	
222	* re_init_spec;	
223	* re_init_spec;	
224	* re_init_spec;	
225	* re_init_spec;	
226	* re_init_spec;	
227	* re_init_spec;	
228	* re_init_spec;	
229	* re_init_spec;	
230	* re_init_spec;	
231	* re_init_spec;	
232	* re_init_spec;	
233	* re_init_spec;	
234	* re_init_spec;	
235	* re_init_spec;	
236	* re_init_spec;	
237	* re_init_spec;	
238	* re_init_spec;	
239	* re_init_spec;	
240	* re_init_spec;	
241	* re_init_spec;	
242	* re_init_spec;	
243	* re_init_spec;	
244	* re_init_spec;	
245	* re_init_spec;	
246	* re_init_spec;	
247	* re_init_spec;	
248	* re_init_spec;	
249	* re_init_spec;	
250	* re_init_spec;	
251	* re_init_spec;	
252	* re_init_spec;	
253	* re_init_spec;	
254	* re_init_spec;	
255	* re_init_spec;	
256	* re_init_spec;	
257	* re_init_spec;	
258	* re_init_spec;	
259	* re_init_spec;	
260	* re_init_spec;	
261	* re_init_spec;	
262	* re_init_spec;	
263	* re_init_spec;	
264	* re_init_spec;	
265	* re_init_spec;	
266	* re_init_spec;	
267	* re_init_spec;	
268	* re_init_spec;	
269	* re_init_spec;	
270	* re_init_spec;	
271	* re_init_spec;	
272	* re_init_spec;	
273	* re_init_spec;	
274	* re_init_spec;	
275	* re_init_spec;	
276	* re_init_spec;	
277	* re_init_spec;	
278	* re_init_spec;	
279	* re_init_spec;	
280	* re_init_spec;	
281	* re_init_spec;	
282	* re_init_spec;	
283	* re_init_spec;	
284	* re_init_spec;	
285	* re_init_spec;	
286	* re_init_spec;	
287	* re_init_spec;	
288	* re_init_spec;	
289	* re_init_spec;	
290	* re_init_spec;	
291	* re_init_spec;	
292	* re_init_spec;	
293	* re_init_spec;	
294	* re_init_spec;	
295	* re_init_spec;	
296	* re_init_spec;	
297	* re_init_spec;	
298	* re_init_spec;	
299	* re_init_spec;	
300	* re_init_spec;	
301	* re_init_spec;	
302	* re_init_spec;	
303	* re_init_spec;	
304	* re_init_spec;	
305	* re_init_spec;	
306	* re_init_spec;	
307	* re_init_spec;	
308	* re_init_spec;	
309	* re_init_spec;	
310	* re_init_spec;	
311	* re_init_spec;	
312	* re_init_spec;	
313	* re_init_spec;	

```

214 1      initargs.timeout = timeout;
215 1      initres = dd_initialize(1)
216 1          /* initargs, handler -> dd_binding_handle );
217 1          /* Will have 1 for RPC call */
218 1
219 1      if (initres == NULL)
220 1          return EP_RB_RECOVER_RPC_FAIL;
221 1
222 1      statergs.service_handle = initres -> service_handle;
223 1      statergs.status = 0;
224 1      statergs = dd_getservicestatus(1)
225 1          /* statergs, handler -> dd_binding_handle );
226 1
227 1      if (statergs == NULL)
228 1          return EP_RB_RECOVER_RPC_FAIL;
229 1
230 1      while (statergs -> status == DD_SERVICE_STARTING)
231 1      {
232 1          time_t now;
233 1
234 1          xdr_free( xdr_DD_getservicestatus_result, (char *)statergs );
235 1          if (now >= end_time)
236 1          {
237 1              rec_apl_log_cm( SUB_CSM_RPC_FAIL,
238 1                  "timeout waiting for ddinitialized to start restore
239 1                  engine" );
240 1              return EP_RB_RECOVER_SERVERFAIL;
241 1          }
242 1
243 1          sleep(1);
244 1          statergs = dd_getservicestatus(1) /* statergs,
245 1              handler -> dd_binding_handle );
246 1
247 1          if (statergs == NULL)
248 1          {
249 1              rec_apl_log_cm( SUB_CSM_RPC_FAIL,
250 1                  "failure getting status from ddinitialized while starting restore
251 1                  engine" );
252 1              return EP_RB_RECOVER_SERVERFAIL;
253 1          }
254 1
255 1          if (statergs -> status != DD_SERVICE_RUNNING)
256 1          {
257 1              rec_apl_log_cm( SUB_CSM_RPC_FAIL,
258 1                  "ddinitialized failure while starting restore
259 1                  engine" );
260 1              return EP_RB_RECOVER_SERVERFAIL;
261 1          }
262 1          return EP_RB_RECOVER_SERVERFAIL;
263 1
264 1          memory( handler -> opaque128,
265 1              sizeof( handler -> opaque128 ) );
266 1          atexit( handler -> opaque128 );
267 1
268 1          xdr_free( xdr_DD_getservicestatus_result, (char *)statergs );
269 1
270 1          xdr_free( xdr_DD_getservicestatus_result, (char *)statergs );
271 1
272 1

```

```

274 1      /* Restore Engine FUNCTIONALITY BEGINS HERE */
275 1      /*
276 1          RE_CLIENT_TSPRC( re_init_args );
277 1          /*
278 1          retval = csc_private_ifspec_init(
279 1              {
280 1                  unsigned char *) handler -> opaque128,
281 1                  re_protonum,
282 1                  re_version,
283 1                  re_ifspec,
284 1                  statergs );
285 1
286 1          if (retval == 0)
287 1          {
288 1              rec_apl_log_cm( SUB_CSM_RPC_FAIL,
289 1                  "failure initializing interface to restore
290 1                  engine" );
291 1              return EP_RB_RECOVER_SERVERFAIL;
292 1          }
293 1
294 1          apl_status = EP_RB_RECOVER_SERVERFAIL;
295 1          do {
296 1              time_t now;
297 1              time_t know;
298 1              if (now >= end_time)
299 1              {
300 1                  rec_apl_log_cm( SUB_CSM_RPC_FAIL,
301 1                      "timeout connecting to restore
302 1                      engine" );
303 1                  return EP_RB_RECOVER_SERVERFAIL;
304 1              }
305 1
306 1              sleep( 1 );
307 1              /* give restore engine time to get going */
308 1              retval = csc_connect_to( unsigned char *) hostname,
309 1                  re_ifspec,
310 1                  RE_CLIENT_GROUP,
311 1                  handler -> dd_binding_handle,
312 1                  statergs );
313 1              if ((status == error_status.OK) && (retval != 0))
314 1              {
315 1                  apl_status = E_SUCCESS;
316 1              } while (apl_status != E_SUCCESS);
317 1
318 1              if (apl_status == E_SUCCESS)
319 1              {
320 1                  re_handle = handler -> re_binding_handle;
321 1                  #ifdef DEBUG
322 1                  /*
323 1                      increase rpc timeout during debugging */
324 1                  rpc_timeout.tv_usec = RPC_TIMEOUT;
325 1                  ctrl_control( re_handle, CMSG_TIMEOUT, (
326 1                      char *)rpc_timeout );
327 1                  re_init_args.username = human_name;
328 1                  set_rpc_obj( re_initialize, are_init_args, RfObjID );
329 1                  re_init_result = re_initialize(
330 1                      re_init_result, are_init_args, re_handle );
331 1                  if ((re_init_result) && are_init_args)
332 1                  {
333 1                      apl_status = EP_RB_RECOVER_RPC_FAIL;
334 1                      rec_apl_log_cm( SUB_CSM_RPC_FAIL,
335 1                          "failure communicating with restore
336 1                          engine" );
337 1                      return EP_RB_RECOVER_SERVERFAIL;
338 1                  }
339 1              }
340 1

```

```

332 2         }
333 3         else {
334 4             api_status = re_init_result->status;
335 5             /* release RPC result struct: */
336 6             xdr_free( xdr_re_status_result, (
337 7                 char *)re_init_result);
338 8         }
339 9     }
340 10     else
341 11     {
342 12         rec_api_log_errm( SUB_CSM_RPC_FAIL,
343 13             "failure connecting to restore engine" );
344 14     }
345 15     if (
346 16         api_status == E_SUCCESS) /* return rest eng handle on success */
347 17     {
348 18         *svrhl = (serverhandle)re_handle;
349 19         return( api_status );
350 20     }
351 21     /* End of EDWRST_initialize() */

```



```

/*
 * Copyright 1997, 1998 EMC Corporation
 */

/*
 * Leading & causes rprocn to pass a line directly through to the output,
 * the ending, ending in a line, this allows the .h to make a little
 * more sense and be properly documented.
 */

/*
 * dispatch_daemon.x : EDM Dispatch Daemon C/S communication module
 */
/*
 * Mission Statement: This is an RPROCEN file which defines the RPC interface
 * between the Dispatch Daemon (which resides on
 * the EDM server) and the backup client callers of its
 * functions. This defines the RPC level calls that a
 * 'caller' can make and the 'service' will respond to.
 */
/*
 * Primary Data Acted On: This defines the data that will flow over the wire.
 * The RPC mechanism will take care of data marshalling
 */

/*
 * Compile-Time Options:
 * This actually gets run through RPROCEN not compiled.
 * It must be run through with the -h flag to create a
 * header, the -m flag to create the service side
 * routines, the -l flag to create the client side
 * routines, and the -c flag to create the common data
 * marshalling routines.
 */
/*
 * Basic Ideas here:
 * Define the RPC level interfaces to the Dispatch Daemon
 * and all data types that will be passed via RPC.
 */
/*
 * Constant Definitions
 */
/*
 * Data Structure Definitions
 */
/*
 * struct DD_rpc_objid
 */
/*
 * int type; /* Object identifier (DD_TYPE) */
#define DD_OPRV_INTV_IN 1 /* Initialize Input Object */
#define DD_OPRV_INTV_OUT 2 /* Initialize Output Object */
/*
 * long len; /* Length of structure, version number */
 */
/*
 * struct DD_client_session_id {
 *     unsigned long high;
 *     unsigned long low;
 * };
 */
/*
 * struct DD_SERVICE_RESTORE=1;
 * // structures for input and output of re_initialize rpc call: */
/*
 * int service;
 */
/*
 * dispatch_daemon.x.1
 */
/*
 * Page 1 of 4
 */
 */
/*
 * struct DD_SERVICE_FAILURE_NONZERO=4;
 * const DD_SERVICE_FAILURE_PRINT=2;
 * const DD_SERVICE_FAILURE_ERROR=1;
 * const DD_SERVICE_STARTING=1;
 * const DD_SERVICE_RUNNING=2;
 * const DD_SERVICE_COMPLETED=4;
 */
/*
 * struct DD_initialize_result {
 *     unsigned int status;
 *     DD_client_session_id service_handle;
 * };
 */
/*
 * structures for getstatus function */
/*
 * struct DD_getserv_status_args {
 *     int status;
 *     DD_client_session_id service_handle;
 * };
 */
/*
 * struct DD_getserv_status_result {
 *     int status;
 *     opaque handle<128>;
 * };
 */
/*
 * // work item type */
/*
 * // These match the throughput for the most part. There are
 * // some extras for identifying NOS workitems.
 */
/*
 * const FS_BACKUP_TYPE = 0;
 * const SHARED_PART_BACKUP_TYPE = 1;
 * const SHARED_PART_BACKUP_TYPE = 2;
 * const SHARED_PART_BACKUP_TYPE = 3;
 * const ONLINE_KICKER_TYPE = 4;
 * const ONLINE_LISTEN_TYPE = 5;
 * const DCONN_KICK_TYPE = 6;
 * const DCONN_NET_TYPE = 7;
 * const DCONN_WMR_TYPE = 8;
 */
/*
 * // length of various buffers */
/*
 * const MESSAGE_SIZE = 6;
 * const TAILLINE_SIZE = 16;
 * const WINLINE_SIZE = 64;
 * const TIKELINE_SIZE = 64;
 * const USERLINE_SIZE = 64;
 * const DCONN_SIZE = 64;
 * const CLIENTLINE_SIZE = 64;
 * const SERVER_SIZE = 256;
 * const MAX_STRING_SIZE = 256;
 */
/*
 * // defines for operation type */
/*
 * const DCONN_SIZE = 64;
 * const RESPONSE_TYPE = 2;
 * const OTHER_TYPE = 16;
 */
/*
 * // work item structure */
/*
 * struct WItem {
 *     time_t started;
 *     unsigned long cur_line;
 *     unsigned long total_bytes_sofar;
 *     total_files;
 * };
 */
/*
 * dispatch_daemon.x.2
 */
/*
 * Page 2 of 4
 */

```



```

CheckSpatchSessions      33      (EJND)spatchSession.cc)
FreeSpatchServerPcodes... 35      (EJND)spatchSession.cc)
FreeSessionInfo          23      (EJND)spatchService.cc)
GetDspatchInfo           38      (EJND)spatchSession.cc)
GetDspatchStatus         36      (EJND)spatchSession.cc)
GetSessionStatus         36      (EJND)spatchSession.cc)
InitializeSession        26      (EJND)spatch.c)
LockSpatchMutex          26      (EJND)spatchSession.cc)
SendPingAsynchronous... 30      (EJND)spatchSession.cc)
UnLockSpatchMutex        27      (EJND)spatchSession.cc)
UpdateSessionLastReceived 33      (EJND)spatchSession.cc)
DaemonBecomeDaemon       10      (EJND)spatch.c)
DaemonCheckInterrupts    5      (EJND)spatch.c)
DaemonCleanup            18      (EJND)spatch.c)
DaemonInitializeLogging... 9      (EJND)spatch.c)
DaemonSessionInitialize... 21      (EJND)spatch.c)
dd.getSessionInfo_L_svc... 22      (EJND)spatchService.cc)
dd.initialize_L_svc..... 20      (EJND)spatchService.cc)
display_uuage            4      (EJND)spatch.c)
kill_handler             3      (EJND)spatch.c)
parse_commandline        3      (EJND)spatch.c)
startSession              4      (EJND)spatchSession.cc)
rpc_init                  12      (EJND)spatch.c)
rpc_run                   15      (EJND)spatch.c)

```

```

ZmDspatch.c ..... 1
daemon ..... 2
daemon_become_daemon ..... 10
daemon_catch_interrupts ..... 5
daemon_check_proper_id ..... 7
daemon_cleanup ..... 18
daemon_daemonize ..... 9
daemon_specific_initialization ..... 16
display_usage ..... 4
kill_handler ..... 3
parse_commandline ..... 8
ipc_init ..... 12
ipc_listen ..... 15
ZmDspatchService.c ..... 19
PresSessionInfo ..... 23
dd_getservicestatus_l_svc ..... 21
dd_sessioninfo_l_svc ..... 22
dd_initialize_l_svc ..... 20
ZmDspatchSessionInfo.c ..... 25
daemonize ..... 31
DnsSessionDescriptors ..... 33
GetDspatchInfo ..... 40
GetDspatchStatus ..... 38
getsessionstatus ..... 36
initialize_session ..... 32
send_ping_to_session ..... 26
SendPingToSession ..... 30
UnlockSessionMutex ..... 27
UpdateSessionLastSent ..... 31
removeSession ..... 43
```


Mon Oct 13 16:05:00 2008	kill_handler	Page 3 of 44
99	/*..... *****	
100	/* Routine: kill_handler	
101	/*	
102	/* Inputs: int signal - the signal which was received.	
103	/*	
104	/* Outputs: Will log messages telling what action is being taken.	
105	/*	
106	/* Return Codes:	
107	/*	
108	/* exits with the number of the signal received	
109	/*	
110	/* Purpose: This routine handles specific signals i.e. SIGINT,	
111	/* SIGQUIT, SIGTERM. Each results in a log entry and an exit.	
112	/*	
113	/* Intended caller: Internal only.	
114	/*..... *****	
115	/*	
116	static void kill_handler (IN int signal)	
117	{	
118	int	
119	struct t_line;	
120	char *ctimebuf;	
121	char	
122	/*buff;	
123	/*	
124	/* If main exits, it calls this routine with signal 0 */	
125	/*	
126	/* Unregister the interface */	
127	(void) csc_unregister_server_interface(&it_spec, &status);	
128	/*	
129	/* If the unregister fails, report the problem, but continue */	
130	if (status != error_status_ok)	
131	{	
132	/*buff = (char *) csc_get_error(status);	
133	/*	
134	(void) EDMDispatch_logout(
135	/*FILE, _LINE, LOG_ERR, MESSAGE_NO_LOGIN, 0,	
136	/*CSC_SERVER_LOGIN_FAILED, <td> &*/,	
137	status, (/*buff ? /*buff : "Unknown error"));	
138	}	
139	/* Get the current time */	
140	(void) time(¤t_time);	
141	/*	
142	ctimebuf = ctime(¤t_time);	
143	/*	
144	/* Overlay newline with null - but should always be 26 bytes long */	
145	ctimebuf[strlen(ctimebuf) - 1] = 0;	
146	/*	
147	(void) EDMDispatch_logout(
148	/*FILE, _LINE, LOG_INFO, MESSAGE_SHUTDOWN, 0,	
149	/*Shutting down as its due to signal &td;, signal);	
150	/*	
151	/* Removes our lock file.	
152	/*	
153	(void) EndDestroyFile(&IDPATH);	
154	/*	
155	exit(signal);	
156	/*	
157	/* End of kill_handler() */	

Mon Oct 13 16:05:00 2008	display_usage	Page 4 of 44
159	/*..... *****	
160	/* Function Name:	
161	/* display_usage	
162	/*	
163	/* Simply displays the usage	
164	/*	
165	/* Call Arguments:	
166	/* Program name	
167	/*	
168	/* Error Outputs and Side Effects:	
169	/* Prints usage.	
170	/*	
171	/* Special Considerations:	
172	/* None.	
173	/*	
174	/*..... *****	
175	/*	
176	static void	
177	display_usage (IN char *programme)	
178	{	
179	/* Print out usage stmc. */	
180	fprintf (stderr, "Usage: %s [-d]\n", programme);	
181	fprintf (
182	stderr, " -d keep the daemon from forking so debugging is easier\n");	
183	/* End display_usage() */	

```

187  /*
188  **
189  ** Routine: daemon_catch_interrupts
190  **
191  ** Inputs:      None
192  **
193  ** Outputs:     None
194  **
195  ** Return Codes: None
196  **
197  ** Purpose:     Sets up signals for service. On NT we will have to
198  **              consider what OS constructs to replace signals with.
199  **              In this case we are catching SIGTERM, SIGINT, and
200  **              SIGQUIT and ignoring anything else.
201  **
202  **              Intended caller: internal only.
203  **
204  **
205  ****
206  */
207 void daemon_catch_interrupts()
208 {
209     struct sigaction  actions;      /* signal actions */
210     ZERO( actions );
211
212     /*
213     ** Set an empty list so we can set signals we want to handle
214     **
215     ** (void) sigemptyset( &actions.sa_mask );
216     */
217
218     /*
219     ** Add signals that we want to handle
220     **
221     ** (void) sigaddset( &actions.sa_mask, SIGTERM );
222     ** (void) sigaddset( &actions.sa_mask, SIGINT );
223     ** (void) sigaddset( &actions.sa_mask, SIGQUIT );
224     */
225     /* Setup the signal handler. */
226     actions.sa_handler = KillHandler;
227
228     /*
229     ** Assign handler to each signal we are interested in.
230     **
231     ** (void) sigaction( SIGTERM, &actions, NULL );
232     ** (void) sigaction( SIGINT, &actions, NULL );
233     ** (void) sigaction( SIGQUIT, &actions, NULL );
234     */
235     /*
236     ** Setup mask so we can specify what signals we will ignore.
237     **
238     ** (void) sigfillset( &actions.sa_mask );
239     */
240     /*
241     ** We want to ignore everything except those we have set up
242     ** above so remove those from the list.
243     **
244     ** (void) sigdelset( &actions.sa_mask, SIGTERM );
245     ** (void) sigdelset( &actions.sa_mask, SIGINT );
246     ** (void) sigdelset( &actions.sa_mask, SIGQUIT );
247     */
248 }

```

```

249 1
250 1
251 1
252 1
253 }

```

* Set the mask. Since no other threads have been started,
 * all threads will get this mask.
 */
 (void) pthread_sigsetmask(SIG_SETMASK, &actions.sa_mask, NULL);

```

256 /*****
257 **
258 ** Routine: daemon_check_proper_ID
259 **
260 ** Inputs:      None
261 ** Outputs:     None
262 **
263 ** Return Codes: exits with an error when the user is not root
264 **
265 ** Purpose:      Checks user's ID and determines if the user is allowed
266 **               to execute service.
267 **               If there are no constraints then this
268 **               function may be blank.
269 **
270 ** Intended caller: Internal only.
271 **
272 *****/
273
274 void daemon_check_proper_ID()
275 {
276     /*
277     ** Check for root
278     */
279     if (geteuid() != E_ROOTUID)
280     {
281         (void) EDMDispatch_logent(
282             "FILE:____LINE____, LOG_ERR, DAEMON_NOTSUPERUSER, 0,
283             "Must be run as superuser, uid was %d",
284             geteuid());
285         exit(1);
286     }
287 }
288
289 }

```

```

291 /*****
292 **
293 ** Routine: parse_commandline
294 **
295 ** Inputs:      argc, argv (command line arguments)
296 **
297 ** Outputs:     None
298 **
299 ** Return Codes: exits with an error when the user types a bad argument
300 **
301 ** Purpose:      Parses command line arguments and sets flags. If there
302 **               are no flags to be set then this function may be empty.
303 **
304 ** Intended caller: Internal only.
305 **
306 *****/
307
308 void parse_commandline(int argc, char *argv[])
309 {
310     int
311     :
312     :
313     :
314     :
315     while ((opt = getopt(argc,argv,"d*")) != EOF )
316     {
317         switch(opt)
318         {
319             case 'd':
320                 g_debug = TRUE;
321                 break;
322             default:
323                 (void) display_usage( argv[0] );
324                 exit(1);
325         }
326     }
327 }
328
329 }
330
331 }

```



```

332  /*.....
333  **
334  ** Routine: daemon_initialize_logging
335  **
336  ** Inputs:
337  **         None
338  **
339  ** Outputs:
340  **         None
341  **
342  ** Return Codes:
343  **         None
344  **
345  ** Purpose:
346  **         Do whatever it takes to initialize logging.
347  **         In the near
348  **         future this may involve doing something with catalogs
349  **         or
350  **         calling higher level logging functions which
351  **         encapsulate
352  **         these things.
353  **
354  ** Intended caller: Internal only.
355  **
356  **.....
357  */
358
359 void
360 daemon_initialize_logging()
361 {
362     /* Pass in argv(0), the program name */
363     (void) esl_log_init(commandlineargs[0]);
364 }

```

```

360  /*.....
361  **
362  ** Routine: daemon_become_daemon
363  **
364  ** Inputs:
365  **         None
366  **
367  ** Outputs:
368  **         None
369  **
370  ** Return Codes:
371  **         exits with an error code if initialization fails
372  **
373  ** Purpose:
374  **         This function is for doing the forking etc. under UNIX.
375  **         It is unknown what will be necessary under NT.
376  **
377  ** Intended caller: Internal only.
378  **.....
379  */
380
381 void
382 daemon_become_daemon()
383 {
384     char *ptr;
385     int ret = 0;
386
387     /* Strip the path from the program name so we can use it
388     * elsewhere.
389     */
390     ptr = strrchr(commandlineargs[0], '/');
391     if (ptr == NULL)
392         ptr = commandlineargs[0];
393     else
394         ptr++;
395
396     /* Change directory to a process specific core directory */
397     ret = esl_coredir_setup(ptr);
398     if (ret != 0)
399     {
400         (void) EDMDspatch_logerrn( _FILE_, _LINE_, LOG_ERR,
401             "MESSAGE_ERR_IN_ESL_CORDIR", 0,
402             "esl_coredir_setup failed: %s",
403             errno );
404     }
405     exit(1);
406 }
407
408 /*
409  ** This is now esl functionality.
410  ** This code does everything necessary
411  ** to make this a "real" daemon by detaching from the
412  ** terminal
413  ** Changing the process group, closing stdout, stderr, stdin,
414  **
415  */
416
417 if (0, debug == FALSE)
418 {
419     ret = esl_daemon_startup();
420     if (ret != 0)
421     {
422         fprintf(
423             stderr, "%s: Failed to initialize as daemon.\n",
424             commandlineargs[0]);
425     }
426 }

```


Mon Oct 13 16:06:00 2008	Mon Oct 13 16:06:00 2008
mpc_init	mpc_init
Page 13 of 44	Page 14 of 44
<pre> 488 3 (void) EMDDispatch_logent(489 3 __FILE__, __LINE__, LOG_ERR, 490 3 MESSAGE_NO_LOGIN, 0, 491 3 "CSC_SERVER_LOGIN failed: <ts> 492 3 status { 493 3 ebuf ? ebuf : "Unknown error"; 494 3 } 495 3 } 496 3 } 497 3 /* If the failure was due to unavailable client, 498 3 * pause and then try again. 499 3 */ 500 3 if (status == sec_roy_server_unavailable) 501 3 { 502 3 /* 503 3 * uses sleep when SUNRPC, otherwise uses 504 3 * pthread call to delay for the specified 505 3 * time 506 3 */ 507 3 CSC_SLEEP(sleep_interval); 508 3 continue; 509 3 } 510 3 /* If we got here, we had a unexpected failure. */ 511 3 (void) EMDDispatch_logent(__FILE__, __LINE__, LOG_ERR, 512 3 MESSAGE_NO_LOGIN, 0, 513 3 "The service cannot log in as 514 3 required"); 515 3 exit(1); 516 3 } 517 3 518 3 /* 519 3 * We need to initialize the authorization module before we 520 3 * do 521 3 */ 522 3 ** a listen. 523 3 /* 524 3 * (void)csc_authorization_init(status); 525 3 */ 526 3 if (status != error_status.ok) 527 3 { 528 3 ebuf = (char *) csc_get_error(status); 529 3 (void) EMDDispatch_logent(__FILE__, __LINE__, LOG_ERR, 530 3 MESSAGE_NONAUTHORIZATION, 0, 531 3 "CSC_AUTHORIZATION_INIT failed: <ts> %s", 532 3 status, { 533 3 ebuf ? ebuf : "Unknown error"); 534 3 exit(1); 535 3 } 536 3 537 3 (void) csc_register_server_interface(&lt;ts> spec, 538 3 SERVER_ANNOTATION, 539 3 status); 540 3 541 3 if (status != error_status.ok) 542 3 { 543 3 ebuf = (char *) csc_get_error(status); 544 3 (void) EMDDispatch_logent(__FILE__, __LINE__, LOG_ERR, 545 3 MESSAGE_CANNOTREGISTER, 0, 546 3 "CSC_REGISTER_SERVER_INTERFACE failed: 547 3 <ts> %s", 548 3 status, { 549 3 ebuf ? ebuf : "Unknown error"); 550 3 exit(1); 551 3 } </pre>	<pre> 541 3 exit(1); 542 3 } 543 3 } 544 3 545 3 </pre>
Mon Oct 13 16:06:00 2008	Mon Oct 13 16:06:00 2008
EMDDispatch.c 13	EMDDispatch.c 14
Page 13 of 44	Page 14 of 44

```

545 /*****
546 **
547 ** Routing: rpc_run
548 ** Inputs:  None
549 ** Outputs: None
550 **
551 ** Return Codes:
552 **
553 ** Purpose:
554 ** This function is for running the RPC listen.
555 ** This is pretty standard between UNIX and NT.
556 **
557 ** Intended caller: Internal only.
558 **
559 **
560 **
561 *****/
562 */
563 void rpc_run()
564 {
565     error_status_t status;
566     /* error status (base.h) */
567     char *buff;
568     /* listen for RPC calls forever. */
569     (void) csc_server_listen(
570         rpc_c_listen_max_calls_default, &status);
571     while (char *) csc_get_error( status );
572     /* We don't expect to get here. */
573     (void) EXDDispatch_logent( LOG_ERR, MESSAGE_SERVER_LISTEN, 0,
574         "FILE: __FILE__, LOG_SERVER_LISTEN failed: %s",
575         status );
576     /* Unknown error */
577     }

```

```

360 /*****
361 **
362 ** Routing: daemon_specific_initialization
363 ** Inputs:  None
364 ** Outputs: None
365 **
366 ** Return Codes:
367 **
368 ** Purpose:
369 ** Do whatever makes this daemon special.
370 ** In some cases you
371 ** may want to start a thread or open a socket.
372 ** Do that here.
373 **
374 ** Intended caller: Internal only.
375 **
376 **
377 *****/
378 void
379 daemon_specific_initialization()
380 {
381     error_status_t status;
382     /* error status (base.h) */
383     int
384         thread_id;
385     pthread_t
386         thread;
387     char
388         *current_time;
389     struct rlimit
390         rlimit;
391     /* Create a file and lock it so we don't start multiple
392     ** daemons. Exit if there is another copy of us running.
393     ** The CreatePidFile call already logs errors so just exit.
394     */
395     if ( !CreatePidFile(PIDPATH) )
396     {
397         exit(1);
398     }
399     /* Find out what time it is */
400     (void) time(&current_time);
401     /* Overlay newline with null - but should always be 26 bytes
402     long */
403     current_time = time(&current_time);
404     climbbuf[ strlen(climbbuf) - 1 ] = 0;
405     /* Log startup message */
406     (void) EXDDispatch_logent(
407         LOG_INFO, MESSAGE_STARTUP, 0,
408         "FILE: __FILE__, Restore service to starting up at %s",
409         current_time );
410     /* set the open files limit very large */
411     rlp_rlim_max = FD_SETSIZE;
412     rlp_rlim_cur = FD_SETSIZE;
413     setrlimit(RLIMIT_NOFILE, &rlp);

```



```

1  *
2  *
3  * Copyright 1996, 1997 EMC Corporation
4  */
5
6  // EMDDispatchService.c
7  *
8  *
9  * Mission Statement:  RPC entry points.
10 *
11 * Primary Data Acctd On:
12 *
13 * Compile-Time Options:
14 *
15 * Basic Ids here:
16 */
17
18 #if defined(linux)
19 #define RCS_id [] = "e(1)SRCSfile: EMDDispatchService.c.v $ "
20 #define RCS_revision "1.1.5"
21 #define State: "1997/02/06 20:49:15 $" ;
22
23 #endif
24
25 #include <cs/compat.h>
26
27 #include <logging/logging.h>
28
29 #include <cs/crc/crc.h>
30
31 #include <restore/dispatch_daemon.h>
32
33 #include <EMDDispatchLog.h>
34 #include <EMDDispatchSession.h>
35
36 /*
37  * These are all the RPC entry points for the dispatch daemon.
38  * The dispatch daemon is multi-threaded and it is the main thread
39  * which handles all incoming RPC.  OMC RPC is single threaded
40  * so each call blocks other RPC calls.  This provides us some
41  * safety in the way we handle our data and limits our exposure
42  * to unexpected multi-threading problems.
43  */
44
45 static void FreeSessionInfo(SessionInfo *);
46
47 //*****
48
49 **
50 ** Routine:  dd_initialize.1
51 **
52 ** Inputs:  DD_initialize_args * - args for the restore initialize
53 **
54 ** Outputs:  None
55 **
56 ** Return Codes:
57 **      DD_initialize_result * - result of init function call
58 **
59 ** Purpose:  Function to create a restore session.
60 **
61 ** Intended caller:  Internal Only.
62 **
63 DD_initialize_result *

```

```

73  /*****
74  **
75  ** Routine: dd_getservicestatus_1
76  **
77  ** Inputs: DD_getservicestatus_args * - args for the
78  **          getservicestatus call
79  **
80  ** Outputs: None
81  **
82  ** Return Codes:
83  **              DD_getservicestatus_result * - result of status function
84  **              call
85  **
86  ** Purpose: Function to poll for status on a session.
87  **
88  ** Intended caller: Internal only.
89  **
90  **
91  */
92  DD_getservicestatus_result *
93  dd_getservicestatus_1_svc(
94      IN DD_getservicestatus_args *arg, IN struct svc_req *req )
95  {
96      static DD_getservicestatus_result argzz;
97
98      GetDispatchStatus(arg, &argzz);
99
100     return &argzz;

```

```

100  /*****
101  **
102  ** Routine: dd_getsessioninfo_1
103  **
104  ** Inputs: DD_getservicestatus_args * - args for the getsessioninfo
105  **          call
106  **
107  ** Outputs: None
108  **
109  ** Return Codes:
110  **              SessionBlock * - result of session info call
111  **
112  ** Purpose: Function to get information on all sessions.
113  **
114  ** Intended caller: Internal only.
115  **
116  **
117  */
118  SessionBlock *
119  dd_getsessioninfo_1_svc(
120      IN DD_getservicestatus_args *arg, IN struct svc_req *req )
121  {
122      static SessionBlock argzz;
123      static bool session_info_first = TRUE;
124
125      if (!first)
126      {
127          memset(&argzz, 0, sizeof(argzz));
128          first = FALSE;
129      }
130      else
131      {
132          FreeSessionInfo(&argzz, &sess);
133          argzz.sess = NULL;
134      }
135
136      GetDispatchInfo(arg, &argzz);
137
138     return &argzz;

```

```

139 /*****
140 **
141 ** Routine: FreeSessionInfo
142 **
143 ** Inputs: SessionInfo * - arg to Free
144 **
145 ** Outputs: None
146 **
147 ** Return Codes:
148 **      None
149 **
150 ** Purpose: Function to free all SessionInfo structures in a list.
151 **
152 ** Intended caller: Internal only.
153 *****/
154
155 static void FreeSessionInfo(SessionInfo *sess)
156 {
157     if (sess == NULL)
158         return;
159
160     if (sess->next != NULL)
161         FreeSessionInfo(sess->next);
162     free(sess);
163 }
164

```



```

1  //
2  // Copyright 1996, 1999 EMC Corporation
3  //
4  //
5  // EDMDispatchSession.cc
6  //
7  // Mission Statement: This is where all session management occurs.
8  //
9  // Primary Data Accessed On:
10 //
11 //
12 // Compile-Time Options:
13 //
14 //
15 // USE_SMRPC - Compile source with smrpc
16 // not set, assume DCS support. If
17 //
18 // Basic idea here: Module for session management
19 //
20 //
21 // The following provides an RCS id in the binary that can be located
22 // with the whot(1) utility. The intent is to keep this short.
23 //
24 // $Id: dispatchSession.cc,v 5.0
25 // $Date: 1997/02/06 20:49:15 $
26 //
27 //
28 //
29 //
30 // #define _POSIX_SOURCE  unable to compile with this define set
31 // #define _POSIX_SOURCE  unable to compile with this define set
32 //
33 #include <asi/c_portable.h>
34 #include <asi/ep_xopen.h>
35 #include <asi/inout.h>
36 #include <thread.h>
37 #include <memory.h>
38 #include <sys/time.h>
39 #include <sys/types.h>
40 #include <syslog.h>
41 //
42 // Define how includes
43 #include <ev/collect.h>
44 #include <ev/cvile.h>
45 #include <ev/vetram.h>
46 #include <ev/bintree.h>
47 //
48 #include <ev/occomm.h>
49 #include <ev/restore/dispatch_daemon.h>
50 #include <ev/restore/dispatch_protocol_client.h>
51 #include <EDMSession.h>
52 #include <EDMRetrievalSession.h>
53 #include <EDMDispatchAgent.h>
54 #include <EDMDispatchConfig.h>
55 #include <EDMDispatchConfig.h>
56 #include <EDMDispatchConfig.h>
57 #include <EDMDispatchConfig.h>
58 #include <EDMDispatchConfig.h>
59 #include <EDMDispatchConfig.h>
60 #include <EDMDispatchConfig.h>
61 #include <EDMDispatchConfig.h>
62 #include <EDMDispatchConfig.h>
63 #include <EDMDispatchConfig.h>
64 #include <EDMDispatchConfig.h>

```

```

65 static int maxDisconnectionTime = SECONDS_PER_HOUR; // one hour
66 //
67 //
68 //
69 //
70 //
71 //
72 //
73 //
74 //
75 //
76 //
77 //
78 //
79 //
80 //
81 //
82 //
83 //
84 //
85 //
86 //
87 //
88 //
89 //
90 //
91 //
92 //
93 //
94 //
95 //

```

```

98  /*****
99  **
100  ** Routine: DllUnlockSessionMutex
101  **
102  ** Inputs: None
103  **
104  ** Outputs: None
105  **
106  ** Return Codes:
107  **      None
108  ** Purpose: Unlock the mutex for the session tree object
109  **
110  **
111  *****/
112
113 static void
114 DllUnlockSessionMutex()
115 {
116     pthread_mutex_unlock(&g_sessionTreeMutex);
117 }
118

```

```

120  /*****
121  **
122  ** Routine: InitializeSession
123  **
124  ** Inputs:   DD_initialize_args *arg - args sent via RPC for starting
125  **          struct svc_req *req - the request block from RPC session
126  **
127  ** Outputs:  DD_initialize_result *res - the result structure which
128  **          operation succeeded or failed.
129  **
130  ** Return Codes:
131  **      None
132  ** Purpose: Initialize a session for the GUI.
133  **
134  *****/
135
136 void
137 InitializeSession(IN DD_initialize_args *arg, IN struct svc_req *req,
138                  OUT DD_initialize_result *res)
139 {
140     EDMSession *session;
141     EDMSession *ret;
142     pthread_t id;
143     time_t c;
144
145     if (arg == NULL || req == NULL || res == NULL)
146     {
147         return;
148     }
149
150     c = time(NULL);
151
152     session = new EDMSession();
153
154     if (session == NULL)
155     {
156         res->status = DD_SERVICE_FAILURE_NONEXC;
157         return;
158     }
159
160     session->InitSession();
161
162     session->SetStartTime(c);
163
164     session->SetOperationType(arg->service);
165
166     if (arg->username != NULL && arg->hostname != NULL)
167     {
168         switch(arg->service)
169         {
170             // code is commented out because we do not
171             // want to read the config for permission information
172             // at this time, it is a waste of cycles
173             case DD_SERVICE_RESORE : boolean,ly allowed;
174
175             allowed =
176

```

```

181 }
182
183 {
184     if (!allowed)
185         res -> status = DD_SERVICE_FAILURE_PERMS;
186     delete session;
187     return;
188 }
189
190 break;
191
192 default: // Add some error message for unknown service
193     break;
194 }
195
196 else
197 {
198     res -> status = DD_SERVICE_FAILURE_NONEXC;
199     delete session;
200     return;
201 }
202
203 LockSessionMutex();
204 ret = (EMSession *) G_sessionTree.Insect({
205     RMCollectable *, session);
206
207 UnlockSessionMutex();
208
209 if (ret == NULL)
210 {
211     res -> status = DD_SERVICE_FAILURE_NONEXC;
212     delete session;
213     return;
214 }
215
216 session -> getSessionID(&ret -> service_handle);
217
218 // Call Steve's thread
219 pthread_create(&id, NULL, &DDService_Init, (void *) session);
220 session -> setThreadID(id);
221
222 return;
223 }
224

```

```

226
227 //*****
228 ** Routine: SendPingMessageToSession
229 ** Inputs: None
230 ** Outputs: None
231 ** Return Codes: None
232 ** Purpose: Queue up all the ping messages to the sessions.
233             If they don't respond they should be considered dead.
234             *****
235
236 */
237 void
238 SendPingMessageToSession()
239 {
240     EMMSession *sess;
241     LockSessionMutex();
242
243     RBinaryTreeIterator *sessionIterator = new RBinaryTreeIterator(
244         G_sessionTree);
245     while ( sessionIterator != NULL &&
246            { sess = (EMSession*) (*sessionIterator)() != NULL } )
247     {
248         DD_client_session_id sid;
249         rpc_binding_handle_t *cecb = NULL;
250         int status;
251         ret;
252
253         if (sess -> getSession() != DD_SERVICE_RUNNING)
254             continue;
255
256         sess -> getSessionID(&sid);
257         ret = GetCSCHandle(&id, &cecb, &status);
258         if (ret != 0 || cecb == NULL || *cecb == NULL)
259             continue;
260         PubResponseMessage(&id, ping_request, sid, cecb, &status);
261     }
262
263 // through with iterator
264 if (sessionIterator != NULL)
265     delete sessionIterator;
266
267 UnlockSessionMutex();
268
269 }
270

```

```

232  /*****
233  **
234  ** Routine: UpdateSessionLastReceived
235  ** Inputs:  ID_client_session_id *sessionId - session that sent us
236  **          something
237  ** Outputs: None
238  **
239  ** Return Codes:
240  ** 0 on success and non-zero otherwise
241  **
242  ** Purpose: Update the specified session with the latest received
243  **          message
244  **
245  ** time.
246  *****/
247  */
248
249  int
250  UpdateSessionLastReceived(ID_client_session_id *sessionId)
251  {
252      time_t last = time(NULL);
253      EXMSession *session;
254      EXMSession *ret;
255
256      session = new EXMSession();
257
258      if (sessionId == NULL)
259      {
260          EXMDispatchLogent{
261              __FILE__, __LINE__, LOG_ERR, SESSION_NO_MEMORY, 0,
262              "Failure to create a session block";
263          }
264          return -1;
265      }
266      session -> sessionId(sessionId);
267
268      LockSessionMutex();
269
270      ret = (EXMSession *) G_sessionTree.find((PmwCollectable *) session);
271      UnlockSessionMutex();
272
273      delete session;
274
275      if (ret == NULL)
276      {
277          EXMDispatchLogent{
278              __FILE__, __LINE__, LOG_ERR, SESSION_LOOKUP_FAILED, 0,
279              "Failure to update session %ld: id received
280              time",
281              sessionId -> high, sessionId -> low);
282          }
283          return -1;
284      }
285      ret -> setLastReceived(last);
286      return 0;
287  }

```

```

338  /*****
339  **
340  ** Routine: UpdateSessionLastSent
341  ** Inputs:  ID_client_session_id *sessionId - session that sent us
342  **          something
343  ** Outputs: None
344  **
345  ** Return Codes:
346  ** 0 on success and non-zero otherwise
347  **
348  ** Purpose: Update the specified session with the latest sent
349  **          message
350  **
351  ** time.
352  *****/
353  */
354
355  int
356  UpdateSessionLastSent(ID_client_session_id *sessionId)
357  {
358      time_t last = time(NULL);
359      EXMSession *session;
360      EXMSession *ret;
361
362      session = new EXMSession();
363
364      if (sessionId == NULL)
365      {
366          EXMDispatchLogent{
367              __FILE__, __LINE__, LOG_ERR, SESSION_NO_MEMORY, 0,
368              "Failure to create a session block";
369          }
370          return -1;
371      }
372      session -> sessionId(sessionId);
373
374      LockSessionMutex();
375
376      ret = (EXMSession *) G_sessionTree.find((PmwCollectable *) session);
377      UnlockSessionMutex();
378
379      delete session;
380
381      if (ret == NULL)
382      {
383          EXMDispatchLogent{
384              __FILE__, __LINE__, LOG_ERR, SESSION_LOOKUP_FAILED, 0,
385              "Failure to update session %ld: id sent
386              time",
387              sessionId -> high, sessionId -> low);
388          }
389          return -1;
390      }
391      ret -> setLastSent(last);
392      return 0;
393  }

```

```

394  /*.....
395  **
396  ** Routine: CheckDispatchSessions
397  **
398  ** Inputs:  None
399  **
400  ** Outputs: None
401  **
402  ** Return Codes:
403  **      None
404  **
405  ** Purpose: Look for dead sessions and kill them off
406  **
407  **.....
408
409  */
410  void
411  CheckDispatchSessions()
412  {
413      EDMSession *seas;
414      int
415      status = 0;
416      int
417      ret = 0;
418      time_t
419      curTime;
420      RtnBinaryTree reapertree;
421      curTime = time(NULL);
422
423      LockSessionMutex();
424      RtnBinaryTreeIterator *seasontree = new RtnBinaryTreeIterator(
425          &seasontree);
426      while ( (seas = EDMSession*)(*seasontree()) != NULL ) {
427          if ( (seas->getIsReceived()) != 0 ) {
428              <= curTime - maxDisconnectionTime && seas->getIsReceived() != 0 ||
429              (seas->getCurTime() <= curTime - maxDisconnectionTime &&
430              seas -> getStatut(
431                  == DD_SERVICE_FAILURE_KNOWN || seas -> getStatut(
432                      == DD_SERVICE_FAILURE_PENDING ) ) )
433              {
434                  // Insert it into the reapertree
435                  (void) reapertree.insert(seas);
436              }
437          }
438          // through with iterator
439          if (seasontree != NULL)
440              delete seasontree;
441          delete seasontree;
442          UnlockSessionMutex();
443
444          // If the reapertree has something in it then use those entries
445          // things from the query tree
446          if (reapertree.entries() > 0)
447              seasontree = new RtnBinaryTreeIterator(reapertree);
448
449  }

```

```

453  2      while (seasontree != NULL &&
454  3              (seas = EDMSession*)(*seasontree()) != NULL ) {
455  4          ID_ClientSession id seasID;
456  5          seas -> getSeasID(seasID);
457  6          ret = removeSeasID(seasID, seasID);
458  7          if (ret != 0)
459  8              EDMDispatchLogent(
460  9                  FILE, LOG_ERR, 0, 0,
461  10                     "Removing session %ld", seasID, seasID, seasID);
462  11          continue;
463  12      }
464  13      else
465  14      {
466  15          EDMDispatchLogent(
467  16              FILE, LOG_INFO, 0, 0,
468  17              "Removing session %ld", seasID, seasID, seasID);
469  18          Haven't received anything since %ld. Current %ld",
470  19              seasID, seasID, seasID, seasID);
471  20          curTime - maxDisconnectionTime;
472  21          ret = deleteHandler(seasID, seasID, seasID);
473  22          if (ret != 0)
474  23          {
475  24              EDMDispatchLogent(
476  25                  FILE, LOG_ERR, 0, 0,
477  26                  "Failure to delete handler for session %ld", seasID, seasID, seasID);
478  27          }
479  28          // through with iterator
480  29          if (seasontree != NULL)
481  30              delete seasontree;
482  31          delete seasontree;
483  32          reapertree.clear();
484  33      }
485  34
486  35  }

```

```

495 /*****
496 ** Routine: DrdInssesAndDescriptors
497 **
498 ** Inputs:  None
499
500 ** Outputs: None
501
502 ** Return Codes:
503 504      None
505
506 ** Purpose: Drain whatever data is on stdout and stderr for sessions.
507
508 *****/
509
510 void
511 DrdInssesAndDescriptors()
512 {
513     int  hour = 0, herr = 0, status = 0;
514     int  select = 0;
515     int  i = 0;
516     char  buff[1024];
517     struct timeval timetowait = {
518         1, 0
519     };
520     fd_set stdoutSet;
521     fd_set stderrSet;
522
523     getStdoutSet(&stdoutSet, &hour, &status);
524
525     if ( (select = select(
526         {
527             For (i = hour+1; i++)
528                 if (FD_ISSET(i, &stdoutSet))
529                     while (read(i, buff, 1024) > 0);
530         }
531     )
532     )
533     {
534         getStderrSet(&stderrSet, &hour, &status);
535
536         if ( (select = select(
537             {
538                 herr = 1, &stderrSet, NULL, NULL, &timetowait) > 0)
539             {
540                 For (i = 0; i < herr+1; i++)
541                     if (FD_ISSET(i, &stderrSet))
542                         while (read(i, buff, 1024) > 0);
543             }
544         }
545     }
546 }

```

```

610 2      *status = SESSION_LOOKUP_FAILED;
612 2      return 1;
613 1      }
615 1      *status = ret -> getStatus();
617 1      return 0;
618 1      }

```

```

620      /*****
621      ** Routine: GetDispatchStatus
622      ** Inputs:  ID_GetServiceStatus_args *arg - session ID to check the
623      **           ID_GetServiceStatus_result *res - the result structure
624      **           status of
625      **
626      ** Outputs: ID_GetServiceStatus_result *res - the result structure
627      **           whether operation succeeded or failed.
628      **
629      ** Return Codes:
630      **               None
631      **
632      ** Purpose: Get status on the starting session.
633      *****/
634      void
635      GetDispatchStatus(IN ID_GetServiceStatus_args *arg,
636                       OUT ID_GetServiceStatus_result *res)
637      {
638          EDMSession *sees;
639          static char buff(CONNECT_HANDLE_SIZE);
640          sees = new EDMSession();
641          if (sees == NULL)
642              // Give an error
643              EDMSessionLogent(
644                  _FILE_, _LINE_, LOG_ERR, SESSION_NO_MEMORY, 0,
645                  "Failure to create a session block.");
646          return;
647      }
648      sees -> setSessionID(karg -> service_handle);
649      LockSessionMutex();
650      ret = (EDMSession *) Q_sessionTree.find((PmcCollectable *) sees);
651      UnlockSessionMutex();
652      delete sees;
653      if (ret == NULL)
654      {
655          EDMSessionLogent(
656              _FILE_, _LINE_, LOG_ERR, SESSION_LOOKUP_FAILED, 0,
657              "Failure to lookup session %d:%d",
658              arg -> service_handle.hsh,
659              arg -> service_handle.low);
660      }
661      res -> status = DD_SERVICE_FAILURE_NONEEXC;
662      return;
663      res -> status = ret -> getStatus();
664      memcpy(buff, 0, sizeof(buff));
665  }

```

Mon Oct 13 16:06:00 2008	GetDispatchStatus	Page 39 of 44
678 1	if (res -> status == DD_SERVICE_RUNNING)	
680 2	{	
681 2	res -> handle_handle_val = (char *) ret -> getConnectionHandle(
682 1	res -> handle_handle_val = CONNECT_HANDLE_SIZE;	
683 1	};	
684 1	else	
685 2	{	
686 2	res -> handle_handle_val = (char *) buff;	
687 1	res -> handle_handle_val = CONNECT_HANDLE_SIZE;	
688 1	}	

Mon Oct 13 16:06:00 2008	GetDispatchInfo	Page 40 of 44
689	*****	
691	*****	
692	*****	
693	*****	
694	*****	
695	*****	
696	*****	
697	*****	
698	*****	
699	*****	
700	*****	
701	*****	
702	*****	
703	*****	
704	*****	
705	*****	
706	*****	
707	*****	
708	*****	
709	*****	
710	*****	
711	*****	
712	*****	
713	*****	
714	*****	
715	*****	
716	*****	
717	*****	
718	*****	
719	*****	
720	*****	
721	*****	
722	*****	
723	*****	
724	*****	
725	*****	
726	*****	
727	*****	
728	*****	
729	*****	
730	*****	
731	*****	
732	*****	
733	*****	
734	*****	
735	*****	
736	*****	
737	*****	
738	*****	
739	*****	
740	*****	
741	*****	
742	*****	
743	*****	
744	*****	
745	*****	
746	*****	
747	*****	
748	*****	
749	*****	
750	*****	
751	*****	
752	*****	
753	*****	
754	*****	
755	*****	
756	*****	
757	*****	
758	*****	
759	*****	
760	*****	
761	*****	
762	*****	
763	*****	
764	*****	
765	*****	
766	*****	
767	*****	
768	*****	
769	*****	
770	*****	
771	*****	
772	*****	
773	*****	
774	*****	
775	*****	
776	*****	
777	*****	
778	*****	
779	*****	
780	*****	
781	*****	
782	*****	
783	*****	
784	*****	
785	*****	
786	*****	
787	*****	
788	*****	
789	*****	
790	*****	
791	*****	
792	*****	
793	*****	
794	*****	
795	*****	
796	*****	
797	*****	
798	*****	
799	*****	
800	*****	
801	*****	
802	*****	
803	*****	
804	*****	
805	*****	
806	*****	
807	*****	
808	*****	
809	*****	
810	*****	
811	*****	
812	*****	
813	*****	
814	*****	
815	*****	
816	*****	
817	*****	
818	*****	
819	*****	
820	*****	
821	*****	
822	*****	
823	*****	
824	*****	
825	*****	
826	*****	
827	*****	
828	*****	
829	*****	
830	*****	
831	*****	
832	*****	
833	*****	
834	*****	
835	*****	
836	*****	
837	*****	
838	*****	
839	*****	
840	*****	
841	*****	
842	*****	
843	*****	
844	*****	
845	*****	
846	*****	
847	*****	
848	*****	
849	*****	
850	*****	
851	*****	
852	*****	
853	*****	
854	*****	
855	*****	
856	*****	
857	*****	
858	*****	
859	*****	
860	*****	
861	*****	
862	*****	
863	*****	
864	*****	
865	*****	
866	*****	
867	*****	
868	*****	
869	*****	
870	*****	
871	*****	
872	*****	
873	*****	
874	*****	
875	*****	
876	*****	
877	*****	
878	*****	
879	*****	
880	*****	
881	*****	
882	*****	
883	*****	
884	*****	
885	*****	
886	*****	
887	*****	
888	*****	
889	*****	
890	*****	
891	*****	
892	*****	
893	*****	
894	*****	
895	*****	
896	*****	
897	*****	
898	*****	
899	*****	
900	*****	
901	*****	
902	*****	
903	*****	
904	*****	
905	*****	
906	*****	
907	*****	
908	*****	
909	*****	
910	*****	
911	*****	
912	*****	
913	*****	
914	*****	
915	*****	
916	*****	
917	*****	
918	*****	
919	*****	
920	*****	
921	*****	
922	*****	
923	*****	
924	*****	
925	*****	
926	*****	
927	*****	
928	*****	
929	*****	
930	*****	
931	*****	
932	*****	
933	*****	
934	*****	
935	*****	
936	*****	
937	*****	
938	*****	
939	*****	
940	*****	
941	*****	
942	*****	
943	*****	
944	*****	
945	*****	
946	*****	
947	*****	
948	*****	
949	*****	
950	*****	
951	*****	
952	*****	
953	*****	
954	*****	
955	*****	
956	*****	
957	*****	
958	*****	
959	*****	
960	*****	
961	*****	
962	*****	
963	*****	
964	*****	
965	*****	
966	*****	
967	*****	
968	*****	
969	*****	
970	*****	
971	*****	
972	*****	
973	*****	
974	*****	
975	*****	
976	*****	
977	*****	
978	*****	
979	*****	
980	*****	
981	*****	
982	*****	
983	*****	
984	*****	
985	*****	
986	*****	
987	*****	
988	*****	
989	*****	
990	*****	
991	*****	
992	*****	
993	*****	
994	*****	
995	*****	
996	*****	
997	*****	
998	*****	
999	*****	
1000	*****	

Mon Oct 13 16:06:00 2008	GetDispatchInfo	Page 41 of 44
768 2	res -> sess = (SessionInfo *) calloc(1, sizeof(SessionInfo));	
769 2	if (res -> sess == NULL)	
770 2	{	
771 2	EDMDispatch_Logent(
772 2	__FILE__, "Failure to allocate session info	
773 2	block");	
774 2	return;	
775 2	}	
776 2	sinfo = res -> sess;	
777 2	ret -> getSessionId(sinfo -> service_handle);	
778 2	sinfo -> status = ret -> getStatus();	
779 2	sinfo -> operation_time = ret -> getStartTime();	
780 2	sinfo -> operation_type = ret -> getOperationType();	
781 2	sinfo -> lastReceived = ret -> getLastReceived();	
782 2	sinfo -> lastReceived = ret -> getLastReceived();	
783 2	else	
784 2	{	
785 2	res -> totalSessions = 0;	
786 2	res -> sess = (SessionInfo *) calloc(1, sizeof(SessionInfo));	
787 2	if (res -> sess == NULL)	
788 2	{	
789 2	EDMDispatch_Logent(
790 2	__FILE__, "Failure to allocate session info	
791 2	block");	
792 2	return;	
793 2	}	
794 2	sinfo = res -> sess;	
795 2	RbBinaryTreeIterator *sessionIterator = new	
796 2	RbBinaryTreeIterator(sinfo->sessInfoTree);	
797 2	bool* lv admaxc = FALSE;	
798 2	while (sessionIterator != NULL && (ret = (EMSession*) {	
799 2	while (sessionIterator != NULL && (*sessionIterator()) != NULL) {	
800 2	int	
801 2	status;	
802 2	if (admaxc)	
803 2	{	
804 2	sinfo -> next = (SessionInfo *) calloc(1, sizeof(
805 2	SessionInfo));	
806 2	if (sinfo -> next == NULL)	
807 2	{	
808 2	break;	
809 2	}	
810 2	sinfo = sinfo -> next;	
811 2	}	
812 2	ret -> getSessionId(sinfo -> service_handle);	
813 2	sinfo -> status = ret -> getStatus();	
814 2	sinfo -> operation_time = ret -> getStartTime();	
815 2	sinfo -> operation_type = ret -> getOperationType();	
816 2	sinfo -> lastReceived = ret -> getLastReceived();	
817 2	sinfo -> lastReceived = ret -> getLastReceived();	
818 2	getHandleSet(
819 2	sinfo -> service_handle, sinfo -> outHandle,	
820 2	sinfo -> lastReceived, sinfo -> lastReceived);	
821 2	res -> totalSessions++;	
822 2	sinfo -> next = NULL;	
823 2	admaxc = TRUE;	
824 2	} // through with iterator	
825 2	if (sessionIterator != NULL)	
826 2	{	
827 2	delete sessionIterator;	
828 2	}	
829 2	unlockSessionMutex();	
830 2	}	

Mon Oct 13 16:06:00 2008	GetDispatchInfo	Page 42 of 44
831 2	res -> sess = (SessionInfo *) calloc(1, sizeof(SessionInfo));	
832 2	if (res -> sess == NULL)	
833 2	{	
834 2	EDMDispatch_Logent(
835 2	__FILE__, "Failure to allocate session info	
836 2	block");	
837 2	return;	
838 2	}	
839 2	sinfo = res -> sess;	
840 2	ret -> getSessionId(sinfo -> service_handle);	
841 2	sinfo -> status = ret -> getStatus();	
842 2	sinfo -> operation_time = ret -> getStartTime();	
843 2	sinfo -> operation_type = ret -> getOperationType();	
844 2	sinfo -> lastReceived = ret -> getLastReceived();	
845 2	sinfo -> lastReceived = ret -> getLastReceived();	
846 2	else	
847 2	{	
848 2	res -> totalSessions = 0;	
849 2	res -> sess = (SessionInfo *) calloc(1, sizeof(SessionInfo));	
850 2	if (res -> sess == NULL)	
851 2	{	
852 2	EDMDispatch_Logent(
853 2	__FILE__, "Failure to allocate session info	
854 2	block");	
855 2	return;	
856 2	}	
857 2	sinfo = res -> sess;	
858 2	RbBinaryTreeIterator *sessionIterator = new	
859 2	RbBinaryTreeIterator(sinfo->sessInfoTree);	
860 2	bool* lv admaxc = FALSE;	
861 2	while (sessionIterator != NULL && (ret = (EMSession*) {	
862 2	while (sessionIterator != NULL && (*sessionIterator()) != NULL) {	
863 2	int	
864 2	status;	
865 2	if (admaxc)	
866 2	{	
867 2	sinfo -> next = (SessionInfo *) calloc(1, sizeof(
868 2	SessionInfo));	
869 2	if (sinfo -> next == NULL)	
870 2	{	
871 2	break;	
872 2	}	
873 2	sinfo = sinfo -> next;	
874 2	}	
875 2	ret -> getSessionId(sinfo -> service_handle);	
876 2	sinfo -> status = ret -> getStatus();	
877 2	sinfo -> operation_time = ret -> getStartTime();	
878 2	sinfo -> operation_type = ret -> getOperationType();	
879 2	sinfo -> lastReceived = ret -> getLastReceived();	
880 2	sinfo -> lastReceived = ret -> getLastReceived();	
881 2	getHandleSet(
882 2	sinfo -> service_handle, sinfo -> outHandle,	
883 2	sinfo -> lastReceived, sinfo -> lastReceived);	
884 2	res -> totalSessions++;	
885 2	sinfo -> next = NULL;	
886 2	admaxc = TRUE;	
887 2	} // through with iterator	
888 2	if (sessionIterator != NULL)	
889 2	{	
890 2	delete sessionIterator;	
891 2	}	
892 2	unlockSessionMutex();	
893 2	}	

```

820  /*****
831  **
832  ** Routine: removeSession
833  **
834  ** Inputs:
835  **
836  ** Outputs:
837  **
838  ** Return Codes:
839  **
840  **      None
841  **
842  ** Purpose: remove the active session object between the GUI and the
843  **           Service.
844  **
845  **
846  int
847  removeSession(IN PD_CLIENT_SESSION_ID *sess_id,
848                OUT int *status)
849  {
850      EDMSession *sess;
851      EDMSession *ret;
852      if (*status == NULL)
853      {
854          return -1;
855      }
856      if (sess_id == NULL)
857      {
858          *status = SESSION_BAD_ARGS;
859          return -1;
860      }
861      if (*status == NULL)
862      {
863          *status = 0;
864          if (!G_SessionTree.isEmpty())
865          {
866              EDMSession *sess;
867              EDMSession *ret;
868              if (!G_SessionTree.isEmpty())
869              {
870                  *status = SESSION_LOOKUP_FAILED, 0,
871                  sess_id -> high, sess_id -> low;
872                  return -1;
873              }
874              sess = new EDMSession();
875              if (sess == NULL)
876              {
877                  EDMSession *sess;
878                  EDMSession *ret;
879                  if (!G_SessionTree.isEmpty())
880                  {
881                      *status = SESSION_LOOKUP_FAILED, 0,
882                      sess_id -> high, sess_id -> low;
883                      return -1;
884                  }
885                  sess -> removeSession(sess_id);
886                  LockSessionMutex();
887              }
888              ret = (EDMSession *) G_SessionTree.remove(sess);
889          }
890      }
891      return 0;
892  }

```

```

890  1      UnlockSessionMutex();
891  2      if (ret == NULL)
892  3      {
893  4          EDMSession *sess;
894  5          EDMSession *ret;
895  6          if (!G_SessionTree.isEmpty())
896  7          {
897  8              *status = SESSION_LOOKUP_FAILED, 0,
898  9              sess_id -> high, sess_id -> low;
899 10              return -1;
900 11          }
901 12          delete sess;
902 13          delete ret;
903 14          delete sess;
904 15          return 0;
905 16      }
906 17      return 0;

```

D11


```

R$main.o 1
  RunWorkItemRestore..... 5
  R$RSL.Start 2
  RunCleanRestore..... 9
  RunExecutIonOverlidesRestore 11
  RunPreExecRestore..... 7
  alwaysFalse 8
R$main.o 13
  DebugIOFFds 34
  DetermineGlobalDriveUse... 31
  HandleWorkItemRestoreResults 25
  InitiateWorkItemRestore... 23
  InterpretWorkItemRestoreStatus 33
  RunWorkItemRestoreForTrail 29
  Select..... 22
  SendRunItemWorkItemQuit 32
  test_fd..... 35
  test_fd_jump 36
R$main.o 39
  GetAuxprocResults 63
  KillWorkItemRestore..... 65
  QuitWorkItemRestore 61
  StartWorkItemRestore..... 56
  StartUpAuxprocess 51
  generate_cleanupgen..... 51
  read_and_write_cleanupgen_and 52
  reset_recovery_privileges... 60
  set_recovery_privileges 58
  start_cleanupgen..... 44
R$main.o 67
  DemoteAuxChildProc..... 130
  DemoteAuxChildProcStatus 128
  decodecookie..... 106
  do_auxproc 72
  ebr_direct_cmd..... 115
  fd_wait_wait_infr 112
  fd_wait_wait_infr 114
  fd_wait_wait_infr 113
  main..... 71
  parse_remote_slider_info2 122
  rb_getmethod..... 120
  read_CDL_no_sync 110
  remove_slave_reflik..... 108
  remove_slave_reflik 107
  signal_handler..... 107
  write_CDL_no_eintr 111
  z_exec_separate_auxproc... 109
  z_rmefilter 79

```


Mon Oct 13 16:00:37 2008	HNSSL_Script	Page 3 of 134
127 1 128 1 129 1 130 1 131 1 132 1 133 1 134 1 135 1 136 1 137 1 138 1 139 1 140 1 141 1 142 1 143 1 144 1 145 1 146 1 147 1 148 1 149 1 150 1 151 1 152 1 153 1 154 1 155 1 156 1 157 1 158 1 159 1 160 1 161 1 162 1 163 1 164 1 165 1 166 1 167 1 168 1 169 1 170 1 171 1 172 1 173 2 174 2 175 2 176 2 177 2 178 2 179 2 180 2 181 2 182 1	<pre> int ret_pre; int ret_exec; int ret_post; int ret_all_ok; int prepareExit = 0; int cleanExit = 0; bool emm_by_quitFlag = FALSE; char aaction[32]; memset(aaction, 0, 32); (void)time(&rcp->rc_cmd_starttime); (void)time(&rcp->rc_cmd_starttime, aaction, 32); rcp->rc_cmd_last_waftime = rcp->rc_cmd_starttime; REMOVE_NONLINE(aaction); rbe_log_status(0, "Restore Started at %s.", aaction); rbe_log_status(0, "Restore Started by user %s, uid %d, gid %d.", *rcp->rc_backup_app == 0 ? (struct pluginData *){ : (struct pluginData *){ rcp->currentPipr->idData() }-> name); rbe_log_status(0, "Restore Started of %s", *rcp_level_object != template %s, trailset %s", STR_SURE(rcp->rc_top_level_object_name), STR_SURE(rcp->rc_template_name), (rcp->rc_server_thread ? "Alternate": "Primary")); rbe_log_status(0, "Restore Started by user %s, uid %d, gid %d.", STR_SURE(rcp->rc_human_uid, rcp->rc_human_gid[0])); rcp->rc_human_uid, rcp->rc_human_gid[0]); rbe_log_status(0, "Restore Started with client destination %s.", STR_SURE(rcp->rc_client_hostname)); /* If not a network restore, check if plugin has its own start function */ if (rcp->rc_backup_app != 0 && NULL != rcp->currentPipr->pluginArray[pFunctionStartRestore]) { setGlobalStatus(EXMR_STATE_EXECUTE); /* set RP's internal status */ ret_exec = rcp->currentPipr->pFunction[pFunctionStartRestore] (rcp->submitObject, quitTest); if(quitTest()) /* check for abort before return */ rbe_log_status(EP_RP_RECOVER_ABORT, "The restore was quit by the user during execution."); setGlobalStatus(EXMR_STATE_USER_QUIT); /* set RP's internal status */ </pre>	<pre> 183 3 184 2 185 2 186 2 187 2 188 2 189 2 190 2 191 2 192 2 193 2 194 2 195 2 196 2 197 2 198 2 199 2 200 2 201 2 202 2 203 2 204 2 205 2 206 2 207 2 208 2 209 2 210 2 211 2 212 2 213 2 214 2 215 1 216 1 217 1 218 1 219 1 220 1 221 1 222 1 223 1 224 2 225 2 226 2 227 2 228 2 229 2 230 2 231 3 232 3 233 2 234 2 235 2 236 3 237 3 238 3 239 3 240 3 241 3 242 3 243 3 244 3 245 3 246 3 247 3 248 3 249 3 250 3 251 3 252 3 253 3 254 3 255 3 256 3 257 3 258 3 259 3 260 3 261 3 262 3 263 3 264 3 265 3 266 3 267 3 268 3 269 3 270 3 271 3 272 3 273 3 274 3 275 3 276 3 277 3 278 3 279 3 280 3 281 3 282 3 283 3 284 3 285 3 286 3 287 3 288 3 289 3 290 3 291 3 292 3 293 3 294 3 295 3 296 3 297 3 298 3 299 3 300 3 301 3 302 3 303 3 304 3 305 3 306 3 307 3 308 3 309 3 310 3 311 3 312 3 313 3 314 3 315 3 316 3 317 3 318 3 319 3 320 3 321 3 322 3 323 3 324 3 325 3 326 3 327 3 328 3 329 3 330 3 331 3 332 3 333 3 334 3 335 3 336 3 337 3 338 3 339 3 340 3 341 3 342 3 343 3 344 3 345 3 346 3 347 3 348 3 349 3 350 3 351 3 352 3 353 3 354 3 355 3 356 3 357 3 358 3 359 3 360 3 361 3 362 3 363 3 364 3 365 3 366 3 367 3 368 3 369 3 370 3 371 3 372 3 373 3 374 3 375 3 376 3 377 3 378 3 379 3 380 3 381 3 382 3 383 3 384 3 385 3 386 3 387 3 388 3 389 3 390 3 391 3 392 3 393 3 394 3 395 3 396 3 397 3 398 3 399 3 400 3 401 3 402 3 403 3 404 3 405 3 406 3 407 3 408 3 409 3 410 3 411 3 412 3 413 3 414 3 415 3 416 3 417 3 418 3 419 3 420 3 421 3 422 3 423 3 424 3 425 3 426 3 427 3 428 3 429 3 430 3 431 3 432 3 433 3 434 3 435 3 436 3 437 3 438 3 439 3 440 3 441 3 442 3 443 3 444 3 445 3 446 3 447 3 448 3 449 3 450 3 451 3 452 3 453 3 454 3 455 3 456 3 457 3 458 3 459 3 460 3 461 3 462 3 463 3 464 3 465 3 466 3 467 3 468 3 469 3 470 3 471 3 472 3 473 3 474 3 475 3 476 3 477 3 478 3 479 3 480 3 481 3 482 3 483 3 484 3 485 3 486 3 487 3 488 3 489 3 490 3 491 3 492 3 493 3 494 3 495 3 496 3 497 3 498 3 499 3 500 3 501 3 502 3 503 3 504 3 505 3 506 3 507 3 508 3 509 3 510 3 511 3 512 3 513 3 514 3 515 3 516 3 517 3 518 3 519 3 520 3 521 3 522 3 523 3 524 3 525 3 526 3 527 3 528 3 529 3 530 3 531 3 532 3 533 3 534 3 535 3 536 3 537 3 538 3 539 3 540 3 541 3 542 3 543 3 544 3 545 3 546 3 547 3 548 3 549 3 550 3 551 3 552 3 553 3 554 3 555 3 556 3 557 3 558 3 559 3 560 3 561 3 562 3 563 3 564 3 565 3 566 3 567 3 568 3 569 3 570 3 571 3 572 3 573 3 574 3 575 3 576 3 577 3 578 3 579 3 580 3 581 3 582 3 583 3 584 3 585 3 586 3 587 3 588 3 589 3 590 3 591 3 592 3 593 3 594 3 595 3 596 3 597 3 598 3 599 3 600 3 601 3 602 3 603 3 604 3 605 3 606 3 607 3 608 3 609 3 610 3 611 3 612 3 613 3 614 3 615 3 616 3 617 3 618 3 619 3 620 3 621 3 622 3 623 3 624 3 625 3 626 3 627 3 628 3 629 3 630 3 631 3 632 3 633 3 634 3 635 3 636 3 637 3 638 3 639 3 640 3 641 3 642 3 643 3 644 3 645 3 646 3 647 3 648 3 649 3 650 3 651 3 652 3 653 3 654 3 655 3 656 3 657 3 658 3 659 3 660 3 661 3 662 3 663 3 664 3 665 3 666 3 667 3 668 3 669 3 670 3 671 3 672 3 673 3 674 3 675 3 676 3 677 3 678 3 679 3 680 3 681 3 682 3 683 3 684 3 685 3 686 3 687 3 688 3 689 3 690 3 691 3 692 3 693 3 694 3 695 3 696 3 697 3 698 3 699 3 700 3 701 3 702 3 703 3 704 3 705 3 706 3 707 3 708 3 709 3 710 3 711 3 712 3 713 3 714 3 715 3 716 3 717 3 718 3 719 3 720 3 721 3 722 3 723 3 724 3 725 3 726 3 727 3 728 3 729 3 730 3 731 3 732 3 733 3 734 3 735 3 736 3 737 3 738 3 739 3 740 3 741 3 742 3 743 3 744 3 745 3 746 3 747 3 748 3 749 3 750 3 751 3 752 3 753 3 754 3 755 3 756 3 757 3 758 3 759 3 760 3 761 3 762 3 763 3 764 3 765 3 766 3 767 3 768 3 769 3 770 3 771 3 772 3 773 3 774 3 775 3 776 3 777 3 778 3 779 3 780 3 781 3 782 3 783 3 784 3 785 3 786 3 787 3 788 3 789 3 790 3 791 3 792 3 793 3 794 3 795 3 796 3 797 3 798 3 799 3 800 3 801 3 802 3 803 3 804 3 805 3 806 3 807 3 808 3 809 3 810 3 811 3 812 3 813 3 814 3 815 3 816 3 817 3 818 3 819 3 820 3 821 3 822 3 823 3 824 3 825 3 826 3 827 3 828 3 829 3 830 3 831 3 832 3 833 3 834 3 835 3 836 3 837 3 838 3 839 3 840 3 841 3 842 3 843 3 844 3 845 3 846 3 847 3 848 3 849 3 850 3 851 3 852 3 853 3 854 3 855 3 856 3 857 3 858 3 859 3 860 3 861 3 862 3 863 3 864 3 865 3 866 3 867 3 868 3 869 3 870 3 871 3 872 3 873 3 874 3 875 3 876 3 877 3 878 3 879 3 880 3 881 3 882 3 883 3 884 3 885 3 886 3 887 3 888 3 889 3 890 3 891 3 892 3 893 3 894 3 895 3 896 3 897 3 898 3 899 3 900 3 901 3 902 3 903 3 904 3 905 3 906 3 907 3 908 3 909 3 910 3 911 3 912 3 913 3 914 3 915 3 916 3 917 3 918 3 919 3 920 3 921 3 922 3 923 3 924 3 925 3 926 3 927 3 928 3 929 3 930 3 931 3 932 3 933 3 934 3 935 3 936 3 937 3 938 3 939 3 940 3 941 3 942 3 943 3 944 3 945 3 946 3 947 3 948 3 949 3 950 3 951 3 952 3 953 3 954 3 955 3 956 3 957 3 958 3 959 3 960 3 961 3 962 3 963 3 964 3 965 3 966 3 967 3 968 3 969 3 970 3 971 3 972 3 973 3 974 3 975 3 976 3 977 3 978 3 979 3 980 3 981 3 982 3 983 3 984 3 985 3 986 3 987 3 988 3 989 3 990 3 991 3 992 3 993 3 994 3 995 3 996 3 997 3 998 3 999 3 1000 3 </pre>

Mon Oct 13 16:00:37 2008	HNSSL_Script	Page 4 of 134
183 3 184 2 185 2 186 2 187 2 188 2 189 2 190 2 191 2 192 2 193 2 194 2 195 2 196 2 197 2 198 2 199 2 200 2 201 2 202 2 203 2 204 2 205 2 206 2 207 2 208 2 209 2 210 2 211 2 212 2 213 2 214 2 215 1 216 1 217 1 218 1 219 1 220 1 221 1 222 1 223 1 224 2 225 2 226 2 227 2 228 2 229 2 230 2 231 3 232 3 233 2 234 2 235 2 236 3 237 3 238 3 239 3 240 3 241 3 242 3 243 3 244 3 245 3 246 3 247 3 248 3 249 3 250 3 251 3 252 3 253 3 254 3 255 3 256 3 257 3 258 3 259 3 260 3 261 3 262 3 263 3 264 3 265 3 266 3 267 3 268 3 269 3 270 3 271 3 272 3 273 3 274 3 275 3 276 3 277 3 278 3 279 3 280 3 281 3 282 3 283 3 284 3 285 3 286 3 287 3 288 3 289 3 290 3 291 3 292 3 293 3 294 3 295 3 296 3 297 3 298 3 299 3 300 3 301 3 302 3 303 3 304 3 305 3 306 3 307 3 308 3 309 3 310 3 311 3 312 3 313 3 314 3 315 3 316 3 317 3 318 3 319 3 320 3 321 3 322 3 323 3 324 3 325 3 326 3 327 3 328 3 329 3 330 3 331 3 332 3 333 3 334 3 335 3 336 3 337 3 338 3 339 3 340 3 341 3 342 3 343 3 344 3 345 3 346 3 347 3 348 3 349 3 350 3 351 3 352 3 353 3 354 3 355 3 356 3 357 3 358 3 359 3 360 3 361 3 362 3 363 3 364 3 365 3 366 3 367 3 368 3 369 3 370 3 371 3 372 3 373 3 374 3 375 3 376 3 377 3 378 3 379 3 380 3 381 3 382 3 383 3 384 3 385 3 386 3 387 3 388 3 389 3 390 3 391 3 392 3 393 3 394 3 395 3 396 3 397 3 398 3 399 3 400 3 401 3 402 3 403 3 404 3 405 3 406 3 407 3 408 3 409 3 410 3 411 3 412 3 413 3 414 3 415 3 416 3 417 3 418 3 419 3 420 3 421 3 422 3 423 3 424 3 425 3 426 3 427 3 428 3 429 3 430 3 431 3 432 3 433 3 434 3 435 3 436 3 437 3 438 3 439 3 440 3 441 3 442 3 443 3 444 3 445 3 446 3 447 3 448 3 449 3 450 3 451 3 452 3 453 3 454 3 455 3 456 3 457 3 458 3 459 3 460 3 461 3 462 3 463 3 464 3 465 3 466 3 467 3 468 3 469 3 470 3 471 3 472 3 473 3 474 3 475 3 476 3 477 3 478 3 479 3 480 3 481 3 482 3 483 3 484 3 485 3 486 3 487 3 488 3 489 3 490 3 491 3 492 3 493 3 494 3 495 3 496 3 497 3 498 3 499 3 500 3 501 3 502 3 503 3 504 3 505 3 506 3 507 3 508 3 509 3 510 3 511 3 512 3 513 3 514 3 515 3 516 3 517 3 518 3 519 3 520 3 521 3 522 3 523 3 524 3 525 3 526 3 527 3 528 3 529 3 530 3 531 3 532 3 533 3 534 3 535 3 536 3 537 3 538 3 539 3 540 3 541 3 542 3 543 3 544 3 545 3 546 3 547 3 548 3 549 3 550 3 551 3 552 3 553 3 554 3 555 3 556 3 557 3 558 3 559 3 560 3 561 3 562 3 563 3 564 3 565 3 566 3 567 3 568 3 569 3 570 3 571 3 572 3 573 3 574 3 575 3 576 3 577 3 578 3 579 3 580 3 581 3 582 3 583 3 584 3 585 3 586 3 587 3 588 3 589 3 590 3 591 3 592 3 593 3 594 3 595 3 596 3 597 3 598 3 599 3 600 3 601 3 602 3 603 3 604 3 605 3 606 3 607 3 608 3 609 3 610 3 611 3 612 3 613 3 614 3 615 3 616 3 617 3 618 3 619 3 620 3 621 3 622 3 623 3 624 3 625 3 626 3 627 3 628 3 629 3 630 3 631 3 632 3 633 3 634 3 635 3 636 3 637 3 638 3 639 3 640 3 641 3 642 3 643 3 644 3 645 3 646 3 647 3 648 3 649 3 650 3 651 3 652 3 653 3 654 3 655 3 656 3 657 3 658 3 659 3 660 3 661 3 662 3 663 3 664 3 665 3 666 3 667 3 668 3 669 3 670 3 671 3 672 3 673 3 674 3 675 3 676 3 677 3 678 3 679 3 680 3 681 3 682 3 683 3 684 3 685 3 686 3 687 3 688 3 689 3 690 3 691 3 692 3 693 3 694 3 695 3 696 3 697 3 698 3 699 3 700 3 701 3 702 3 703 3 704 3 705 3 706 3 707 3 708 3 709 3 710 3 711 3 712 3 713 3 714 3 715 3 716 3 717 3 718 3 719 3 720 3 721 3 722 3 723 3 724 3 725 3 726 3 727 3 728 3 729 3 730 3 731 3 732 3 733 3 734 3 735 3 736 3 737 3 738 3 739 3 740 3 741 3 742 3 743 3 744 3 745 3 746 3 747 3 748 3 749 3 750 3 751 3 752 3 753 3 754 3 755 3 756 3 757 3 758 3 759 3 760 3 761 3 762 3 763 3 764 3 765 3 766 3 767 3 768 3 769 3 770 3 771 3 772 3 773 3 774 3 775 3 776 3 777 3 778 3 779 3 780 3 781 3 782 3 783 3 784 3 785 3 786 3 787 3 788 3 789 3 790 3 791 3 792 3 793 3 794 3 795 3 796 3 797 3 798 3 799 3 800 3 801 3 802 3 803 3 804 3 805 3 806 3 807 3 808 3 809 3 810 3 811 3 812 3 813 3 814 3 815 3 816 3 817 3 818 3 819 3 820 3 821 3 822 3 823 3 824 3 825 3 826 3 827 3 828 3 829 3 830 3 831 3 832 3 833 3 834 3 835 3 836 3 837 3 838 3 839 3 840 3 841 3 842 3 843 3 844 3 845 3 846 3 847 3 848 3 849 3 850 3 851 3 852 3 853 3 854 3 855 3 856 3 857 3 858 3 859 3 860 3 861 3 862 3 863 3 864 3 865 3 866 3 867 3 868 3 869 3 870 3 871 3 872 3 873 3 874 3 875 3 876 3 877 3 878 3 879 3 880 3 881 3 882 3 883 3 884 3 885 3 886 3 887 3 888 3 889 3 890 3 891 3 892 3 893 3 894 3 895 3 896 3 897 3 898 3 899 3 900 3 901 3 902 3 903 3 904 3 905 3 906 3 907 3 908 3 909 3 910 3 911 3 912 3 913 3 914 3 915 3 916 3 917 3 918 3 919 3 920 3 921 3 922 3 923 3 924 3 925 3 926 3 927 3 928 3 929 3 930 3 931 3 932 3 933 3 934 3 935 3 936 3 937 3 938 3 939 3 940 3 941 3 942 3 943 3 944 3 945 3 946 3 947 3 948 3 949 3 950 3 951 3 952 3 953 3 954 3 955 3 956 3 957 3 958 3 959 3 960 3 961 3 962 3 963 3 964 3 965 3 966 3 967 3 968 3 969 3 970 3 971 3 972 3 973 3 974 3 975 3 976 3 977 3 978 3 979 3 980 3 981 3 982 3 983 3 984 3 985 3 986 3 987 3 988 3 989 3 990 3 991 3 992 3 993 3 994 3 995 3 996 3 997 3 998 3 999 3 1000 3	<pre> return EP_RP_RECOVER_ABORT; } return EP_RP_RECOVER_ABORT; if (E_SUCCESS == ret_exec) setGlobalStatus(EXMR_STATE_FAILED); else setGlobalStatus(EXMR_STATE_SUCCESSFUL); /* set RP's internal status */ return(ret_exec); ret_pre = RunPrepareRestore(submitObject, quitTest, &prepareExit); if(EP_RP_RECOVER_ABORT == ret_pre) { The log statusEP_RP_RECOVER_ABORT, "The restore was quit by the user during preparation."); /* set RP's internal status */ return EP_RP_RECOVER_ABORT; } if(prepareExit != 0) { The log status(EP_RP_RECOVER_ABORT, "The restore failed during preparation. Exit %d", prepareExit); } setGlobalStatus(EXMR_STATE_FAILED); /* set RP's internal status */ return (EP_RP_RECOVER_FAILED); setGlobalStatus(EXMR_STATE_EXECUTE); /* set RP's internal status */ ret_exec = ExecuteWorkItemRestore(submitObject, quitTest); QuitTest = QuitTest(); /* check for abort before cleanup */ if (E_SUCCESS == (ret_all_ok = ret_exec)) /* check if any NFS failed */ if local_stat, EDMSdata status, memset(&stats, 0, sizeof(EDMSstats)); if (0 != getRestoreStatus(0, &stats, &local_stat)) { rbe_log_status("Internal error! Failed in getRestoreStatus."); ret_exec = ret_all_ok = EP_RP_RECOVER_EXECUTEFAILED; } else if (&stats.edm.failed) /* If any workitems failed, it's a failure for cleanup purposes */ if (0 == stats.edm.successfull) ret_all_ok = EP_RP_RECOVER_ALLFAIL; else if (&stats.edm.successfull & &stats.edm.failed) ret_all_ok = EP_RP_RECOVER_MIXEDFAIL; } } </pre>	
Mon Oct 13 16:00:37 2008	HNSSL_Script	Page 4 of 134

Mon Oct 13 16:00:37 2008	RSJSL_Short	Page 5 of 134
<pre> 240 3 else 241 3 rec_allOk = EP_RB_RECOVER_ANYFAIL; 242 3 } 243 1 } 244 1 245 1 rec_post = RunCleanUpReactor(SubnetObjected, 246 1 obj, 247 1 rec_allOk, 248 1 &CleanUpExit); 249 1 250 1 if(QuitFlag) 251 1 { 252 1 rbe_Log_Status(EP_RB_RECOVER_ABORT, 253 1 "The reactor was quit by the user during execution. 254 2 *"); 255 2 secObjStatus(ERMRE_STATE_USER_QUIT); 256 1 return EP_RB_RECOVER_ABORT; 257 1 258 1 } 259 1 if (E_SUCCESS != rec_exec) /* return execute status if it failed 260 2 */ 261 2 { 262 2 secObjStatus(ERMRE_STATE_FAILED); /* sec RE's internal status */ 263 2 return rec_exec; 264 1 } 265 1 if(EP_RB_RECOVER_ABORT == rec_post) 266 1 { 267 2 rbe_Log_Status(EP_RB_RECOVER_ABORT, 268 2 "The reactor was quit by the user during cleanup. *"); 269 2 secObjStatus(ERMRE_STATE_USER_QUIT); /* sec RE's internal status */ 270 1 return EP_RB_RECOVER_ABORT; 271 1 } 272 1 if((CleanUpExit != 0) (E_SUCCESS != rec_post)) 273 1 { 274 2 rbe_Log_Status(EP_RB_RECOVER_POSTFAILED, 275 2 "The reactor failed during cleanup. Exit 64", 276 2 CleanUpExit); 277 2 secObjStatus(ERMRE_STATE_FAILED); /* sec RE's internal status */ 278 2 return (EP_RB_RECOVER_POSTFAILED); 279 2 } 280 1 } 281 1 secObjStatus(ERMRE_STATE_SUCCESSFUL); /* sec RE's internal status */ 282 1 return (E_SUCCESS); 283 1 /* NSTSL_Start */ </pre>		
Mon Oct 13 16:00:37 2008	RSJSLmain.c5	Page 5 of 134

Mon Oct 13 16:00:37 2008	ExecuteWorkItemReactor	Page 6 of 134
<pre> 292 static error_t by 293 ExecuteWorkItemReactor(int SubnetObjected, 294 hostname_by (*QuitTest)(void)) 295 { 296 int rc; 297 RunItem; 298 sm_push(); 299 rc = error_message(0) = 0; 300 if(0 != (rc = RunItem = RunWorkItemReactor(301 SubnetObjected, QuitTest))) 302 { 303 rbe_Log_Status(0, "Internal error: Failed in RunWorkItemReactor"); 304 } 305 sm_pop(); 306 if (QuitTest() == TRUE) 307 return EP_RB_RECOVER_ABORT; 308 if (rc = RunItem != 0) 309 return EP_RB_RECOVER_EXECUTEFAILED; 310 return E_SUCCESS; 311 } </pre>		
Mon Oct 13 16:00:37 2008	RSJSLmain.c6	Page 6 of 134


```

320 #define EXECUTABLE_MAX 1024
321 static errno_t
322 RunPrepareRestore(int SubJobObjcId,
323                  boolean_t (*QuitTest)(void),
324                  int *PrepareExit)
325 {
326     char **prephaseargs = NULL;
327     char **prephaseenv = NULL;
328     int GetStatus = 0;
329     char preexecutable[EXECUTABLE_MAX];
330     boolean_t restore_cancelled = FALSE;
331
332     *PrepareExit = 0;
333
334     /*
335      * GetOPrephase allocates prephaseargs & prephaseenv.
336      * This will need to be free'd later.
337      */
338
339     if (0 != GetOPrephase(SubJobObjcId,
340                           preexecutable,
341                           EXECUTABLE_MAX,
342                           prephaseargs,
343                           prephaseenv,
344                           &GetStatus))
345     {
346         rpe_log_status(0, "Internal error: Failed in GetOPrephase");
347         return (EP_RA_RECOVER_FATALERR);
348     }
349
350     if (0 != attemp(preexecutable, ""))
351     {
352         setGlobalStatus( ERRNE_SWATE_PREPHASE );
353         if (-1 == RunExecutable(FALSE,
354                                 NULL,
355                                 preexecutable,
356                                 prephaseargs,
357                                 prephaseenv,
358                                 prephaseexit,
359                                 &restore_cancelled,
360                                 QuitTest))
361         {
362             rpe_log_status(
363                 0, "Internal error: Failed in RunExecutable for prepare.");
364             return (EP_RA_RECOVER_FATALERR);
365         }
366         if (TRUE == restore_cancelled)
367             return (EP_RA_RECOVER_ABORT);
368     }
369
370     return (E_SUCCESS);
371 }

```

```

375 boolean_t AlwaysFalse() { return FALSE; }

```

Mon Oct 13 16:00:37 2008	RunCleanupFeature	Page 9 of 134
317	static error: by	
318	RunCleanupFeature(int SubmoduleId,	
319	boolean, by ("QuitTest") void),	
320	int CleanupStatus,	
321	int CleanupExit)	
322	{	
323	char *postPhaseArgs = NULL;	
324	char *postPhaseEnv = NULL;	
325	int GetStatus = 0;	
326	char postExecutable[EXECUTABLE_MAX];	
327	boolean, by restore = FALSE;	
328	boolean, by ignore_quit = FALSE;	
329	boolean, by ignore_quit = FALSE;	
330	*CleanupExit = 0;	
331		
332	/*	
333	GetStatusPhase allocates postPhaseArgs & postPhaseEnv.	
334	* This will need to be fixed later.	
335	* This will need to be fixed later.	
336	*/	
337	PutGetStatusQueue();	
338	if (0 != GetStatusPhase(SubmoduleId,	
339	postExecutable,	
340	postPhaseArgs,	
341	postPhaseEnv,	
342	postPhaseArgs,	
343	postPhaseEnv,	
344	GetStatus)	
345	{	
346	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
347	return (EP_RB_RECOVER_FAILURE);	
348	}	
349		
350	/*	
351	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
352	* This will need to be fixed later.	
353	* This will need to be fixed later.	
354	*/	
355	PutGetStatusQueue();	
356	if (0 != GetStatusPhase(SubmoduleId,	
357	postExecutable,	
358	postPhaseArgs,	
359	postPhaseEnv,	
360	postPhaseArgs,	
361	postPhaseEnv,	
362	GetStatus)	
363	{	
364	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
365	return (EP_RB_RECOVER_FAILURE);	
366	}	
367		
368	/*	
369	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
370	* This will need to be fixed later.	
371	* This will need to be fixed later.	
372	*/	
373	PutGetStatusQueue();	
374	if (0 != GetStatusPhase(SubmoduleId,	
375	postExecutable,	
376	postPhaseArgs,	
377	postPhaseEnv,	
378	postPhaseArgs,	
379	postPhaseEnv,	
380	GetStatus)	
381	{	
382	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
383	return (EP_RB_RECOVER_FAILURE);	
384	}	
385		
386	/*	
387	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
388	* This will need to be fixed later.	
389	* This will need to be fixed later.	
390	*/	
391	PutGetStatusQueue();	
392	if (0 != GetStatusPhase(SubmoduleId,	
393	postExecutable,	
394	postPhaseArgs,	
395	postPhaseEnv,	
396	postPhaseArgs,	
397	postPhaseEnv,	
398	GetStatus)	
399	{	
400	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
401	return (EP_RB_RECOVER_FAILURE);	
402	}	
403		
404	/*	
405	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
406	* This will need to be fixed later.	
407	* This will need to be fixed later.	
408	*/	
409	PutGetStatusQueue();	
410	if (0 != GetStatusPhase(SubmoduleId,	
411	postExecutable,	
412	postPhaseArgs,	
413	postPhaseEnv,	
414	postPhaseArgs,	
415	postPhaseEnv,	
416	GetStatus)	
417	{	
418	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
419	return (EP_RB_RECOVER_FAILURE);	
420	}	
421		
422	/*	
423	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
424	* This will need to be fixed later.	
425	* This will need to be fixed later.	
426	*/	
427	PutGetStatusQueue();	
428	if (0 != GetStatusPhase(SubmoduleId,	
429	postExecutable,	
430	postPhaseArgs,	
431	postPhaseEnv,	
432	postPhaseArgs,	
433	postPhaseEnv,	
434	GetStatus)	
435	{	
436	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
437	return (EP_RB_RECOVER_FAILURE);	
438	}	
439		
440	/*	
441	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
442	* This will need to be fixed later.	
443	* This will need to be fixed later.	
444	*/	
445	PutGetStatusQueue();	
446	if (0 != GetStatusPhase(SubmoduleId,	
447	postExecutable,	
448	postPhaseArgs,	
449	postPhaseEnv,	
450	postPhaseArgs,	
451	postPhaseEnv,	
452	GetStatus)	
453	{	
454	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
455	return (EP_RB_RECOVER_FAILURE);	
456	}	
457		
458	/*	
459	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
460	* This will need to be fixed later.	
461	* This will need to be fixed later.	
462	*/	
463	PutGetStatusQueue();	
464	if (0 != GetStatusPhase(SubmoduleId,	
465	postExecutable,	
466	postPhaseArgs,	
467	postPhaseEnv,	
468	postPhaseArgs,	
469	postPhaseEnv,	
470	GetStatus)	
471	{	
472	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
473	return (EP_RB_RECOVER_FAILURE);	
474	}	
475		
476	/*	
477	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
478	* This will need to be fixed later.	
479	* This will need to be fixed later.	
480	*/	
481	PutGetStatusQueue();	
482	if (0 != GetStatusPhase(SubmoduleId,	
483	postExecutable,	
484	postPhaseArgs,	
485	postPhaseEnv,	
486	postPhaseArgs,	
487	postPhaseEnv,	
488	GetStatus)	
489	{	
490	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
491	return (EP_RB_RECOVER_FAILURE);	
492	}	
493		
494	/*	
495	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
496	* This will need to be fixed later.	
497	* This will need to be fixed later.	
498	*/	
499	PutGetStatusQueue();	
500	if (0 != GetStatusPhase(SubmoduleId,	
501	postExecutable,	
502	postPhaseArgs,	
503	postPhaseEnv,	
504	postPhaseArgs,	
505	postPhaseEnv,	
506	GetStatus)	
507	{	
508	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
509	return (EP_RB_RECOVER_FAILURE);	
510	}	
511		
512	/*	
513	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
514	* This will need to be fixed later.	
515	* This will need to be fixed later.	
516	*/	
517	PutGetStatusQueue();	
518	if (0 != GetStatusPhase(SubmoduleId,	
519	postExecutable,	
520	postPhaseArgs,	
521	postPhaseEnv,	
522	postPhaseArgs,	
523	postPhaseEnv,	
524	GetStatus)	
525	{	
526	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
527	return (EP_RB_RECOVER_FAILURE);	
528	}	
529		
530	/*	
531	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
532	* This will need to be fixed later.	
533	* This will need to be fixed later.	
534	*/	
535	PutGetStatusQueue();	
536	if (0 != GetStatusPhase(SubmoduleId,	
537	postExecutable,	
538	postPhaseArgs,	
539	postPhaseEnv,	
540	postPhaseArgs,	
541	postPhaseEnv,	
542	GetStatus)	
543	{	
544	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
545	return (EP_RB_RECOVER_FAILURE);	
546	}	
547		
548	/*	
549	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
550	* This will need to be fixed later.	
551	* This will need to be fixed later.	
552	*/	
553	PutGetStatusQueue();	
554	if (0 != GetStatusPhase(SubmoduleId,	
555	postExecutable,	
556	postPhaseArgs,	
557	postPhaseEnv,	
558	postPhaseArgs,	
559	postPhaseEnv,	
560	GetStatus)	
561	{	
562	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
563	return (EP_RB_RECOVER_FAILURE);	
564	}	
565		
566	/*	
567	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
568	* This will need to be fixed later.	
569	* This will need to be fixed later.	
570	*/	
571	PutGetStatusQueue();	
572	if (0 != GetStatusPhase(SubmoduleId,	
573	postExecutable,	
574	postPhaseArgs,	
575	postPhaseEnv,	
576	postPhaseArgs,	
577	postPhaseEnv,	
578	GetStatus)	
579	{	
580	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
581	return (EP_RB_RECOVER_FAILURE);	
582	}	
583		
584	/*	
585	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
586	* This will need to be fixed later.	
587	* This will need to be fixed later.	
588	*/	
589	PutGetStatusQueue();	
590	if (0 != GetStatusPhase(SubmoduleId,	
591	postExecutable,	
592	postPhaseArgs,	
593	postPhaseEnv,	
594	postPhaseArgs,	
595	postPhaseEnv,	
596	GetStatus)	
597	{	
598	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
599	return (EP_RB_RECOVER_FAILURE);	
600	}	
601		
602	/*	
603	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
604	* This will need to be fixed later.	
605	* This will need to be fixed later.	
606	*/	
607	PutGetStatusQueue();	
608	if (0 != GetStatusPhase(SubmoduleId,	
609	postExecutable,	
610	postPhaseArgs,	
611	postPhaseEnv,	
612	postPhaseArgs,	
613	postPhaseEnv,	
614	GetStatus)	
615	{	
616	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
617	return (EP_RB_RECOVER_FAILURE);	
618	}	
619		
620	/*	
621	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
622	* This will need to be fixed later.	
623	* This will need to be fixed later.	
624	*/	
625	PutGetStatusQueue();	
626	if (0 != GetStatusPhase(SubmoduleId,	
627	postExecutable,	
628	postPhaseArgs,	
629	postPhaseEnv,	
630	postPhaseArgs,	
631	postPhaseEnv,	
632	GetStatus)	
633	{	
634	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
635	return (EP_RB_RECOVER_FAILURE);	
636	}	
637		
638	/*	
639	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
640	* This will need to be fixed later.	
641	* This will need to be fixed later.	
642	*/	
643	PutGetStatusQueue();	
644	if (0 != GetStatusPhase(SubmoduleId,	
645	postExecutable,	
646	postPhaseArgs,	
647	postPhaseEnv,	
648	postPhaseArgs,	
649	postPhaseEnv,	
650	GetStatus)	
651	{	
652	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
653	return (EP_RB_RECOVER_FAILURE);	
654	}	
655		
656	/*	
657	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
658	* This will need to be fixed later.	
659	* This will need to be fixed later.	
660	*/	
661	PutGetStatusQueue();	
662	if (0 != GetStatusPhase(SubmoduleId,	
663	postExecutable,	
664	postPhaseArgs,	
665	postPhaseEnv,	
666	postPhaseArgs,	
667	postPhaseEnv,	
668	GetStatus)	
669	{	
670	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
671	return (EP_RB_RECOVER_FAILURE);	
672	}	
673		
674	/*	
675	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
676	* This will need to be fixed later.	
677	* This will need to be fixed later.	
678	*/	
679	PutGetStatusQueue();	
680	if (0 != GetStatusPhase(SubmoduleId,	
681	postExecutable,	
682	postPhaseArgs,	
683	postPhaseEnv,	
684	postPhaseArgs,	
685	postPhaseEnv,	
686	GetStatus)	
687	{	
688	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
689	return (EP_RB_RECOVER_FAILURE);	
690	}	
691		
692	/*	
693	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
694	* This will need to be fixed later.	
695	* This will need to be fixed later.	
696	*/	
697	PutGetStatusQueue();	
698	if (0 != GetStatusPhase(SubmoduleId,	
699	postExecutable,	
700	postPhaseArgs,	
701	postPhaseEnv,	
702	postPhaseArgs,	
703	postPhaseEnv,	
704	GetStatus)	
705	{	
706	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
707	return (EP_RB_RECOVER_FAILURE);	
708	}	
709		
710	/*	
711	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
712	* This will need to be fixed later.	
713	* This will need to be fixed later.	
714	*/	
715	PutGetStatusQueue();	
716	if (0 != GetStatusPhase(SubmoduleId,	
717	postExecutable,	
718	postPhaseArgs,	
719	postPhaseEnv,	
720	postPhaseArgs,	
721	postPhaseEnv,	
722	GetStatus)	
723	{	
724	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
725	return (EP_RB_RECOVER_FAILURE);	
726	}	
727		
728	/*	
729	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
730	* This will need to be fixed later.	
731	* This will need to be fixed later.	
732	*/	
733	PutGetStatusQueue();	
734	if (0 != GetStatusPhase(SubmoduleId,	
735	postExecutable,	
736	postPhaseArgs,	
737	postPhaseEnv,	
738	postPhaseArgs,	
739	postPhaseEnv,	
740	GetStatus)	
741	{	
742	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
743	return (EP_RB_RECOVER_FAILURE);	
744	}	
745		
746	/*	
747	RestorePhase allocates postPhaseArgs & postPhaseEnv.	
748	* This will need to be fixed later.	
749	* This will need to be fixed later.	
750	*/	
751	PutGetStatusQueue();	
752	if (0 != GetStatusPhase(SubmoduleId,	
753	postExecutable,	
754	postPhaseArgs,	
755	postPhaseEnv,	
756	postPhaseArgs,	
757	postPhaseEnv,	
758	GetStatus)	
759	{	
760	the_log_stats(0, "Internal error: failed in GetStatusPhase");	
761	return (EP_RB_RECOVER_FAILURE);	
762	}	
763		

```

482 static constexpr
483 RunExecutionOverInfiniteStore(int SubscriptId,
484                               bool ann_by (*QuitTest)(void))
485 {
486     return( E_SUCCESS );
487 }

```

```

489 #undef EXECUTABLE_MAX

```



```

/*
 * This sets the number of drives and media access concurrency for
 * i.e. The count of running work item rescues for this trail.
 * Today this is one.
 */
if(0 != SetDrivesAcquired(HighestActiveTrail, 1, &temp_status))
{
    (void)the_user_error(0,
        "Internal error: Cannot set drive acquired;
        1) for trailid %d, cannot continue.", HighestActiveTrail);
    return -1;
}

if (0 > (temp_status = RunWorkItemRescuesForTrail(
    HighestActiveTrail,
    CountDrivesAvailable,
    CancelRescueForTrail,
    &QuitFlag,
    &CountDrivesInUse)))
{
    /* RunWorkItemRescuesForTrail does its own error logging. */
    return -1;
}

if(temp_status == 0)
{
    (void)the_log_states(0,
        "trail %d restore had no work item to run(
        1) *", HighestActiveTrail);
    /* more work may be needed to recover from this error condition. */
}

/* temp_status > 0 */
{
    trailrescuesRunning++;
}

/* And while() initial startup loop */
while(1)
{
    int HighestFD = 0;
    fd_set WorkItemFDs;
    int retStatus;
    struct timeval timeout = {5, 0};

    if(!QuitFlag) && (!SentQuit())
    {
        (void)the_log_states(0,
            "Restore was quit by user. Quitting restore,
            this could take a while.");
        SendQuit = TRUE;
        SendRunningForItemQuit();
    }

    if(0 != getfdset(&WorkItemFDs, &HighestFD, &retStatus))
    {
        (void)the_user_error(0,
            "Internal error: Cannot get auxproc result
            fds, cannot continue.");
    }
}

```

```

return -1;
}

if(0 != debugmode)
{
    DebugLogFds("The file descriptors to wait on are ",
        &WorkItemFDs);
}

if(0 > (retStatus = Select(HighestFD + 1,
    &WorkItemFDs,
    NULL, NULL,
    &timeout)))
{
    /* error */
    (void)the_user_error(HIGHCOVER, USER1(ERROR),
        "Internal error: Cannot get auxproc result
        fds, cannot continue.");
    return -1;
}

else if (0 == retStatus)
{
    /* timed out */
    QuitFlag = CancelRescueForTrail();
}

else
{
    /* Available fds */
    int ReadyFds = retStatus;
    int FoundFds = 0;
    int index;

    if(debugmode)
    {
        DebugLogFds("The file descriptors ready to read are ",
            &WorkItemFDs);
    }

    /* If there are available fds then we may want to
    * schedule the next work item restore. We should
    * check if the user initiated a quit.
    */
    QuitFlag = CancelRescueForTrail();

    for(index = 0;
        (index < (HighestFD + 1)) && (FoundFds < ReadyFds);
        index++)
    {
        int StartWorkItemForTrail = 0;
        if(FD_ISSET(index, &WorkItemFDs))
        {
            int TrailID;
            int TrailAcquired;
            wd_restore_results results;
            FoundFds++;

            mmemcpy(&results, 0, sizeof(wd_restore_results));

            if(0 != HandleWorkItemRescueResults(index,
                &TrailID,
                &results))

```



```

350     epperror
351     InitializeWorkItemRestore(const int SubmitObjID,
352                               const int SubmitElemID)
353     {
354         struct auxproc AuxProcVals;
355         epperror Cy StartUpAuxProcVals = EXIT_FAILURE;
356         time_t StartTime;
357         int TempStatus;
358         char junk_executable[1024];
359         char *xp_fd_from_x;
360         char *xp_gd_from_x;
361         int SocketStatus;
362         char clientName[256] = "";
363         int clientPort;
364         int status;
365         time_t EndTime;
366         /* If there are any environment variables to set.
367          * The restore of the output variables are ignored.
368          */
369         if (E_SUCCESS != GetSocketExecOnPhase(SubmitObjID,
370                                                SubmitElemID,
371                                                &junk_argv,
372                                                &junk_argv,
373                                                &SocketStatus))
374         {
375             (void)the_user_error(0,
376                                   "Internal Error: Could not get environment
377                                   variables.");
378             return -1;
379         }
380         if (E_SUCCESS == GetBERConnConnect(SubmitObjID,
381                                             SubmitElemID,
382                                             clientName, 256,
383                                             &clientPort,
384                                             &SocketStatus))
385         {
386             StartUpAuxResults = StartUpAuxProcess(0 /* XXX */,
387                                                    AuxProcVals,
388                                                    clientName,
389                                                    clientPort);
390             else
391             {
392                 (void)the_user_error(0,
393                                       "Internal Error: Could not get Remote Client name
394                                       &
395                                       port to connect.");
396                 return -1;
397             }
398             if (E_SUCCESS != StartUpAuxResults)
399             {
400                 /* StartUpAuxProcess does its own logging. */
401                 return -1;
402             }
403         }
404         /*
405          * We need to close the bulk fd. This file descriptor
406          * is not being used any more. If we do not close it
407          * here we will have a file descriptor leak because
408          * we won't be able to determine what it was when the
409          * work item completes.
410          */
411     }

```

```

612     /*
613     close(AuxProcVals.xp_fd_bulk_to_x);
614     time(&StartTime);
615     if (0 != newrandseed(AuxProcVals.xp_fd_to_x,
616                          AuxProcVals.xp_fd_from_x,
617                          AuxProcVals.xp_fd_prog_from_x,
618                          SubmitObjID,
619                          SubmitElemID,
620                          AuxProcVals.xp_gd,
621                          KTempStatus))
622     {
623         (void)the_user_error(
624             0, "Internal Error: Could not register handle set.");
625         return -1;
626     }
627     if (0 > StartWorkItemRestore(&cp,
628                                  AuxProcVals,
629                                  SubmitObjID,
630                                  SubmitElemID))
631     {
632         /*
633          * StartWorkItemRestore does logging if initialization fails
634          */
635         (void)the_user_error(
636             0, "Error in StartWorkItemRestore SubmitObjID %d,
637             SubmitElemID %d", SubmitObjID,
638             SubmitElemID);
639         /*
640          * the following code kills auxproc when recc or xcpio do not
641          * start
642          * we do not want an auxproc sitting around.
643          * If errors occur in deleteAndReset or KillWorkItemRestore the
644          * messages are
645          * logged in those calls, plus,
646          * we already know there was an error and that
647          * is why we are doing this right now.
648          */
649         time(&EndTime);
650         deleteAndReset(
651             AuxProcVals.xp_fd_from_x, EndTime, EP_RE_RECOVER_ALLPATH, &status);
652         KillWorkItemRestore(
653             AuxProcVals.xp_gd, AuxProcVals.xp_fd_to_x);
654         return -1;
655     }
656     return 0;
657 }
658 /*
659  */
660 } // InitializeWorkItemRestore() */

```

Mon Oct 13 16:00:37 2008	HandleWorkItemReserveResults	Page 25 of 134
663 664 665 666 667 668	<pre> /* * Interpretate return. * Drain progress. * Get the progress for work item. * Delete the handle sock. */ static int handleWorkItemReserveResults(int fromFD, int trailID, w_reserve_results *results) { int retries = 0; int retries = 0; int GetAuxprocResultsStatus; int GetTempStatus; int GetFromFD; int DrainResult; int wCount; int AuxProcFD; int toFD, getFromFD; time_t EndTime; int timeout = 3; /* Lets try 3 seconds */ boolean rv fromHangup = FALSE; toFD = getFromFD = -1; while(1) {fromHangup}) { GetAuxprocResultsStatus = GetAuxprocResults(fromFD, results); if(!-1 == GetAuxprocResultsStatus) /* GetAuxprocResults() does its own logging */ (void)the_user_error(0, "Error in GetAuxprocResults"); return -1; } if(0 == GetAuxprocResultsStatus) { if(test_fd_hup(fromFD) == 1) { fromHangup = TRUE; } } /* The remote result are not always going to * be set. For example if the remote command * is not started correctly. */ if(results->local_exit_set == TRUE) { break; } else { sleep(1); test_fd(fromFD); continue; } } </pre>	
Mon Oct 13 16:00:37 2008	RSU:mbw/c 13	Page 25 of 134

Mon Oct 13 16:00:37 2008	HandleWorkItemReserveResults	Page 26 of 134
727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785	<pre> time(&endTime); if(0 != PushDrainRequest(fromFD, &tempStatus)) { (void)the_user_error(0, "Internal error: Could not push drain request, cannot continue."); return -1; } /* Lets give the progress thread a chance to drain keeping busy in * the meanwhile. */ if(0 != FindTrailQueueEntry(fromFD, trailID, &tempStatus)) { (void)the_user_error(0, "Internal error: Could not find trail id for finished work item, cannot continue."); return -1; } if(0 != DecrementRunningCnt(*trailID, &wCount, &tempStatus)) { (void)the_user_error(0, "Internal error: Could not decrement running work item for trail, cannot continue."); return -1; } if(0 != getPID(fromFD, &auxProcPid, &tempStatus)) { (void)the_user_error(0, "Internal error: Could not get auxproc pid for work item, cannot continue."); return -1; } if(0 != getHandleDesc(fromFD, &toFD, &getFromFD, &progressFD, &tempStatus)) { (void)the_user_error(0, "Internal error: Could not get auxproc file descriptors for work item, cannot continue."); return -1; } (void)the_user_error(0, "Internal error: mismatch on from file descriptors for work item, cannot continue."); return -1; } while (0 != (ret = PopDrainResult(&timeout, &drainResult, &tempStatus)) && retries < 3) retries++; if (ret != 0) </pre>	
Mon Oct 13 16:00:37 2008	RSU:mbw/c 14	Page 26 of 134

Mon Oct 13 16:00:37 2008	HandleWorkItemReserveResults	Page 27 of 134
796 2	(void) item_user_error(0);	/* Internal error: Could not pop drain results, cannot continue */;
797 2		
798 2		
799 1	return -1;	
799 1	/*	
799 1	Send Final Progress for work item XXX	
799 1	/*	
796 1	/* Translate the local and remote error statuses	
797 1	/* to an operation value:	
798 1	/*	
800 1	if(0 != results->local_exit_status)	/* use local error, if any */
801 1	switch (results->local_exit_status)	
802 2	{	
803 2	case XG_EXIT_ALLFALL:	
804 2	jobstat = EP_RA_RECOVER_ALLFALL;	
805 2	break;	
806 2	case XG_EXIT_MANYFALL:	
807 2	jobstat = EP_RA_RECOVER_MANYFALL;	
808 2	break;	
809 2	case XG_EXIT_FPMFALL:	
810 2	jobstat = EP_RA_RECOVER_FPMFALL;	
811 2	break;	
812 2	case SPEXIN_REMOTE_STDBER_PROTOCOL:	
813 2	jobstat = EP_RA_RECOVER_STDBER_FAIL;	
814 2	jobstat = EP_RA_RECOVER_CLIENT_STDBER_FAIL;	
815 2	case XG_EXIT_STOPPED:	/* treat like signal */
816 2	default:	/* check for signal termination vs all generic failures */
817 2		
818 2	if (XG_EXIT_STOPPED < results->local_exit_status	
819 2	{	
820 2	{ /* XG_EXIT_STOPPED == results->local_exit_status for signal */	
821 2	if (XG_EXIT_STOPPED < SIGPIPE == results->local_exit_status)	
822 3	jobstat = EP_RA_RECOVER_SERVER_SIGPIPE;	
823 3	else	
824 3	jobstat = EP_RA_RECOVER_SERVER_SIGNAL;	
825 2	}	
826 2	else	
827 2	{ /* generic server failure, unless client failed too */	
828 3	jobstat = EP_RA_RECOVER_SERVER_FAIL;	
829 3	if (0 != results->remote_exit_status)	
830 3	jobstat = EP_RA_RECOVER_BOTHFAIL;	
831 2	}	
832 1	else if(0 != results->remote_exit_status)	
833 1	jobstat = EP_RA_RECOVER_CLIENTFAIL;	
834 1	else	
835 1	jobstat = E_SUCCESS;	
836 1		
837 1	if((0 != results->remote_exit_status)	
838 1	((0 != results->local_exit_status))	
839 1	{	
840 2	int asetat=0;	
841 2	int rc=0;	
842 2	char *tempstatname=NULL;	
843 2	char *trallstatname=NULL;	
844 2	rc = gethandlesetInformation(&rcfd,	
845 2	&tempstatname,	
846 2	&trallstatname,	
847 2		
848 2		
849 2		
Mon Oct 13 16:00:37 2008	RSUWsrc 15	Page 27 of 134

Mon Oct 13 16:00:37 2008	HandleWorkItemReserveResults	Page 28 of 134
850 2	errallstatname,	
851 2	status);	
852 2	the_log_stats(0, "Reserve Failure of "	
853 2	top_level Object: %s, template %s.",	
854 2	STR_SURE(tempstatname));	
855 2	free(tempstatname);	
856 2	free(widname);	
857 2	free(trallstatname);	
858 2		
859 1	if(0 != deletehandleset(&rcfd, EndTime, jobstat, &tempstatname))	
860 1	{	
861 2	(void) the_user_error(0,	
862 2	"Internal error: Could not delete handle	
863 2	set, cannot continue.");	
864 2		
865 2	return -1;	
866 2		
867 1	if(0 != killWorkItemReserve(auxprocPid,	
868 1	-1 /* hack this arg is not needed yet	
869 1	and to */))	
870 1	{	
871 2	(void) the_user_error(0,	
872 2	"Internal error: Could not kill finished	
873 2	auxproc, cannot continue.");	
874 2		
875 1	return -1;	
876 1		
877 1	close(&rcfd);	
878 1	close(&rcfd);	
879 1	close(&progrcfd);	
880 1		
881 1	if(debugmode)	
882 2	{	
883 2	(void) the_user_error(0,	
884 2	"DEBUG: HandleWorkItemReserveResults Auxproc(
885 2	PID %d) just finished for %ld id work items left = %d.",	
886 2	auxprocPid,	
887 2	trallid,	
888 2	widcount);	
889 2		
890 2	(void) the_user_error(0,	
891 2	"DEBUG: HandleWorkItemReserveResults Auxproc(
892 2	PID %d) results are local: %d, setPids remote: %d set: %s.",	
893 2	auxprocPid,	
894 2	results -> local_exit_status,	
895 2	results -> remote_exit_status,	
896 2	results -> remote_exit_set ? "TRUE": "FALSE",	
897 2	results -> remote_exit_set ? "TRUE": "FALSE");	
898 1	}	
899 1	return 0;	
900 1	} /* End HandleWorkItemReserveResults() */	
Mon Oct 13 16:00:37 2008	RSUWsrc 16	Page 28 of 134

Mon Oct 13 16:00:37 2008	RunWorkItemRestoresForTrail	Page 29 of 134
904	/*	
905	* RestoreWorkItemRestoresForTrail()	
906	* Description	
907	* This function tears all the work item for the	
908	* trail. For no this is set to one but concurrency	
909	* will can be supported.	
910	* Args:	
911	* (1) TrailID -- the id for this trail.	
912	* (2) CountDrivesAvailable -- the local drives available to restore.	
913	* (3) CountDrivesAvailable -- indicates whether the user has quit the restore.	
914	* (4) QuitFlag -- indicates whether the user has quit the restore.	
915	* (5) CountDrivesInUse -- the count of trails in use by restore.	
916	* Return int	
917	* If 0 then an error has occurred.	
918	* If 0 or greater then the number of trail restores started will be	
919	* returned.	
920	static int	
921	RunWorkItemRestoresForTrail(const int TrailID,	
922	const int CountDrivesAvailable,	
923	boolean by *QuitFlag,	
924	boolean by *CountDrivesInUse)	
925	{	
926	int DriveAcquiredForTrail;	
927	int DriveConcurrencyForTrail;	
928	int subtrailID;	
929	int subtrailElementID;	
930	int popResults = 0;	
931	int temp_status;	
932	int temp_status;	
933	int CountDrivesRestored = 0;	
934	int wCount;	
935	while(1)	
936	{	
937	(*CountDrivesInUse)++;	
938	if(!0 == popResults == PopMTForTrail)Queue(TrailID,	
939	subtrailID,	
940	subtrailElementID,	
941	temp_status)) &&	
942	{	
943	(void)the_user_error(0,	
944	*Internal error: Cannot pop work item off	
945	trail queue, cannot continue.");	
946	return -1;	
947	}	
948	if(!-1 == popResults) && (SCHD.NO_MORE_JOBS == temp_status)	
949	{	
950	return CountOfWorkItemRestoresStarted;	
951	}	
952	temp_status = InitiateWorkItemRestoresForTrail(subtrailID, subtrailElementID);	
953	if(temp_status == 0)	
954	return CountOfWorkItemRestoresStarted;	

Mon Oct 13 16:00:37 2008	RunWorkItemRestoresForTrail	Page 30 of 134
955	{	
956	/* InitiateWorkItemRestoresForTrail() does its own logging */	
957	(void)the_user_error(0, "Error in InitiateWorkItemRestores,"	
958	subtrailID id, subtrailElementID id", subtrailID,	
959	subtrailElementID);	
960	return -1;	
961	}	
962	if(0 == IncrementRunningMT(TrailID, &wCount, &temp_status))	
963	{	
964	(void)the_user_error(0, "Internal error: Could not increment	
965	running work items for trail, cannot continue.");	
966	return -1;	
967	}	
968	CountOfWorkItemRestoresStarted++;	
969	if(0 == GetDrivesAcquiredForTrail,	
970	&DriveAcquiredForTrail,	
971	&temp_status))	
972	{	
973	(void)the_user_error(0, "Internal error: Cannot get drives	
974	acquired, cannot continue.");	
975	return -1;	
976	}	
977	if(0 == GetDriveConcurrencyForTrail,	
978	&DriveConcurrencyForTrail,	
979	&temp_status))	
980	{	
981	(void)the_user_error(0, "Internal error: Cannot get drive	
982	concurrency, cannot continue.");	
983	return -1;	
984	}	
985	if(0 == GetDriveConcurrencyForTrail,	
986	&DriveConcurrencyForTrail,	
987	&temp_status))	
988	{	
989	(void)the_user_error(0, "Internal error: Cannot get drive	
990	concurrency, cannot continue.");	
991	return -1;	
992	}	
993	*QuitFlag = CancelRestoresTest();	
994	if(!DriveAcquiredForTrail < DriveConcurrencyForTrail) &&	
995	(!DriveAcquiredForTrail < CountDrivesAvailable) &&	
996	(FALSE == *QuitFlag))	
997	{	
998	continue;	
999	else	
1000	{	
1001	break;	
1002	}	
1003	return CountOfWorkItemRestoresStarted;	
1004	} /* RunWorkItemRestoresForTrail() */	
1005		
1006		
1007		
1008		
1009		
1010		
1011		
1012		
1013		
1014		

```

1018  /* Scub */
1019  static int DetermineGlobalDriveUse()
1020  {
1021      /* Limiting to MAXINT == not limiting... Need resource management
1022       * to do this properly.
1023       NOTE: This should now work like eh_dc_restore does.
1024       */
1025      return MAXINT;
1026  }

```

```

1028  static int
1029  SendRunningWorkItemsOut()
1030  {
1031      int *aPlist;
1032      int count;
1033      int status;
1034      int index;
1035      if (0 != getpidList(&count, &aPlist, &status))
1036      {
1037          (void)Pda_user_error(0,
1038                              "Internal error: Cannot get auxproc pid list,
1039                              cannot continue.");
1040      }
1041      return -1;
1042  }
1043  for(index = 0; index < count; index++)
1044  {
1045      QuitWorkItemsBefore(aPlist[index]);
1046  }
1047  return 0;
1048  }

```

```

1096  /*
1097  1098  * Stub this out for now.
1099  */
1100  static int
1101  1102  interpretWorkItemPacifierResults(wl_result_t *results)
1103  {
1104  1105  return 0;
1106  }
1107

```

```

1099  static void
1100  1101  DebugLogFds(char *error_msg,
1102  1103  fd_set *fds)
1104  {
1105  int index; fd_count = 0;
1106  char buffer[4096];
1107  char *buffer = (char*)buffer;
1108
1109  for(index=0;
1110  index < 1024;
1111  index++)
1112  {
1113  if (FD_ISSET(index, fds))
1114  {
1115  int size = 0;
1116  size = sprintf(buffer, "%d", index);
1117  buffer += size;
1118  fd_count++;
1119  }
1120  }
1121  if (fd_count > 0)
1122  {
1123  rws_log_stats(0, "fd_count: %d : (%s) %s",
1124  error_msg, fd_count, buffer);
1125  }
1126

```

```

1085 static int
1086 test_fd(int fd)
1087 {
1088     fd_set read_fds;
1089     struct timeval timeout;
1090     struct timeval timeout = { 0, 0 };
1091
1092     FD_ZERO(&read_fds);
1093     FD_SET(fd, &read_fds);
1094
1095     do
1096     {
1097         ret_select = select(fd + 1, &read_fds, NULL, NULL, &timeout);
1098     } while((-1 == ret_select) && (EINVAL == errno));
1099
1100     return ret_select;
1101 }
1102
1103
1104

```

```

1106 /*
1107  * test_fd_hup()
1108  * Description:  map the supplied file descriptor to see if
1109  *               it has had the hang up condition.
1110  * Args:
1111  *   Input fd -- the file descriptor to check for the hang up condition.
1112  * Returns:
1113  *   1 for HUP event received on fd.
1114  *   0 No HUP event received on fd.
1115  *   -1 errno set.
1116  */
1117
1118 static int
1119 test_fd_hup(int fd)
1120 {
1121     struct pollfd fds;
1122     int ret_poll;
1123
1124     if (fd < 0)
1125     {
1126         errno = EINVAL;
1127         return -1;
1128     }
1129
1130     fds.fd = fd;
1131     fds.events = POLLIN;
1132     fds.revents = 0; /* initialize */
1133
1134     do
1135     {
1136         ret_poll = poll(&fds, 1, 0);
1137     } while((-1 == ret_poll) && (EINVAL == errno));
1138
1139     if (-1 == ret_poll)
1140     {
1141         return -1;
1142     }
1143
1144     if (POLLHUP & fds.revents)
1145     {
1146         return 1;
1147     }
1148     else
1149     {
1150         return 0;
1151     }
1152 }
1153
1154 /* end test_fd_hup() */
1155
1156

```



```

122  /* static errno by setup_aux_processes(struct recover_context *)
123  */
124
125  struct debumode,
126  struct auxproc_ernp,
127  char *socket_listen,
128  int clientSocketPort)
129
130
131  {
132
133      /* NOTES:
134       * 1) Do I really want to be reliant on the recover_context struct.
135       * 2) I need to fork() exec() auxproc.
136       * 3) Is fork() really going to work.
137       */
138
139      /*define RFD 0 /* in a pipe, fd 0 is the read descriptor */
140      /*define WRD 1 /* and fd 1 is the write descriptor */
141      int save_errno;
142      int fd;
143      char *resultBuf = NULL;
144      char *auxproc_name = AUPROCNAMNE;
145      char *auxproc_executable = AUPROCNAMNE;
146      int ping_status;
147      int auxproc_index = 0; /* allow Remove */
148      int cmd_pipe_to[2];
149      int cmd_pipe_from[2];
150      int build_pipe_to[2];
151      int build_pipe_from[2];
152      int prog_pipe_from[2];
153
154      if (pipe(cmd_pipe_to) == -1 ||
155          pipe(cmd_pipe_from) == -1 ||
156          pipe(build_pipe_to) == -1 ||
157          pipe(prog_pipe_from) == -1)
158      {
159          save_errno = errno;
160          the_log_stats(RECOVER_MKERR(errno), "pipe() failed");
161          return(RECOVER_MKERR(save_errno));
162      }
163
164      /* This below appends environment variables to
165       * the environment that auxproc inherits from the
166       * restore engine.
167       */
168      if (NULL != auxproc_ernp)
169      {
170          int index;
171          for (index=0; NULL != auxproc_ernp[index]; index++)
172          {
173              if (0 != putenv(auxproc_ernp[index]))
174              {
175                  the_log_stats(RECOVER_MKERR(errno),
176                      "Unable to set auxproc environment %s",
177                      auxproc_ernp[index], getpid());
178              }
179          }
180          _exit(1); /* We are the child */
181      }

```

```

188  1
189  2
190  3
191  4
192  5
193  6
194  7
195  8
196  9
197  10
198  11
199  12
200  13
201  14
202  15
203  16
204  17
205  18
206  19
207  20
208  21
209  22
210  23
211  24
212  25
213  26
214  27
215  28
216  29
217  30
218  31
219  32
220  33
221  34
222  35
223  36
224  37
225  38
226  39
227  40
228  41
229  42
230  43
231  44
232  45
233  46
234  47
235  48
236  49
237  50
238  51
239  52
240  53
241  54
242  55
243  56
244  57
245  58
246  59
247  60
248  61
249  62
250  63
251  64
252  65
253  66
254  67
255  68
256  69
257  70
258  71
259  72
260  73
261  74
262  75
263  76
264  77
265  78
266  79
267  80
268  81
269  82
270  83
271  84
272  85
273  86
274  87
275  88
276  89
277  90
278  91
279  92
280  93
281  94
282  95
283  96
284  97
285  98
286  99
287  100
288  101
289  102
290  103
291  104
292  105
293  106
294  107
295  108
296  109
297  110
298  111
299  112
300  113
301  114
302  115
303  116
304  117
305  118
306  119
307  120
308  121
309  122
310  123
311  124
312  125
313  126
314  127
315  128
316  129
317  130
318  131
319  132
320  133
321  134
322  135
323  136
324  137
325  138
326  139
327  140
328  141
329  142
330  143
331  144
332  145
333  146
334  147
335  148
336  149
337  150
338  151
339  152
340  153
341  154
342  155
343  156
344  157
345  158
346  159
347  160
348  161
349  162
350  163
351  164
352  165
353  166
354  167
355  168
356  169
357  170
358  171
359  172
360  173
361  174
362  175
363  176
364  177
365  178
366  179
367  180
368  181
369  182
370  183
371  184
372  185
373  186
374  187
375  188
376  189
377  190
378  191
379  192
380  193
381  194
382  195
383  196
384  197
385  198
386  199
387  200
388  201
389  202
390  203
391  204
392  205
393  206
394  207
395  208
396  209
397  210
398  211
399  212
400  213
401  214
402  215
403  216
404  217
405  218
406  219
407  220
408  221
409  222
410  223
411  224
412  225
413  226
414  227
415  228
416  229
417  230
418  231
419  232
420  233
421  234
422  235
423  236
424  237
425  238
426  239
427  240
428  241
429  242
430  243
431  244
432  245
433  246
434  247
435  248
436  249
437  250
438  251
439  252
440  253
441  254
442  255
443  256
444  257
445  258
446  259
447  260
448  261
449  262
450  263
451  264
452  265
453  266
454  267
455  268
456  269
457  270
458  271
459  272
460  273
461  274
462  275
463  276
464  277
465  278
466  279
467  280
468  281
469  282
470  283
471  284
472  285
473  286
474  287
475  288
476  289
477  290
478  291
479  292
480  293
481  294
482  295
483  296
484  297
485  298
486  299
487  300
488  301
489  302
490  303
491  304
492  305
493  306
494  307
495  308
496  309
497  310
498  311
499  312
500  313
501  314
502  315
503  316
504  317
505  318
506  319
507  320
508  321
509  322
510  323
511  324
512  325
513  326
514  327
515  328
516  329
517  330
518  331
519  332
520  333
521  334
522  335
523  336
524  337
525  338
526  339
527  340
528  341
529  342
530  343
531  344
532  345
533  346
534  347
535  348
536  349
537  350
538  351
539  352
540  353
541  354
542  355
543  356
544  357
545  358
546  359
547  360
548  361
549  362
550  363
551  364
552  365
553  366
554  367
555  368
556  369
557  370
558  371
559  372
560  373
561  374
562  375
563  376
564  377
565  378
566  379
567  380
568  381
569  382
570  383
571  384
572  385
573  386
574  387
575  388
576  389
577  390
578  391
579  392
580  393
581  394
582  395
583  396
584  397
585  398
586  399
587  400
588  401
589  402
590  403
591  404
592  405
593  406
594  407
595  408
596  409
597  410
598  411
599  412
600  413
601  414
602  415
603  416
604  417
605  418
606  419
607  420
608  421
609  422
610  423
611  424
612  425
613  426
614  427
615  428
616  429
617  430
618  431
619  432
620  433
621  434
622  435
623  436
624  437
625  438
626  439
627  440
628  441
629  442
630  443
631  444
632  445
633  446
634  447
635  448
636  449
637  450
638  451
639  452
640  453
641  454
642  455
643  456
644  457
645  458
646  459
647  460
648  461
649  462
650  463
651  464
652  465
653  466
654  467
655  468
656  469
657  470
658  471
659  472
660  473
661  474
662  475
663  476
664  477
665  478
666  479
667  480
668  481
669  482
670  483
671  484
672  485
673  486
674  487
675  488
676  489
677  490
678  491
679  492
680  493
681  494
682  495
683  496
684  497
685  498
686  499
687  500
688  501
689  502
690  503
691  504
692  505
693  506
694  507
695  508
696  509
697  510
698  511
699  512
700  513
701  514
702  515
703  516
704  517
705  518
706  519
707  520
708  521
709  522
710  523
711  524
712  525
713  526
714  527
715  528
716  529
717  530
718  531
719  532
720  533
721  534
722  535
723  536
724  537
725  538
726  539
727  540
728  541
729  542
730  543
731  544
732  545
733  546
734  547
735  548
736  549
737  550
738  551
739  552
740  553
741  554
742  555
743  556
744  557
745  558
746  559
747  560
748  561
749  562
750  563
751  564
752  565
753  566
754  567
755  568
756  569
757  570
758  571
759  572
760  573
761  574
762  575
763  576
764  577
765  578
766  579
767  580
768  581
769  582
770  583
771  584
772  585
773  586
774  587
775  588
776  589
777  590
778  591
779  592
780  593
781  594
782  595
783  596
784  597
785  598
786  599
787  600
788  601
789  602
790  603
791  604
792  605
793  606
794  607
795  608
796  609
797  610
798  611
799  612
800  613
801  614
802  615
803  616
804  617
805  618
806  619
807  620
808  621
809  622
810  623
811  624
812  625
813  626
814  627
815  628
816  629
817  630
818  631
819  632
820  633
821  634
822  635
823  636
824  637
825  638
826  639
827  640
828  641
829  642
830  643
831  644
832  645
833  646
834  647
835  648
836  649
837  650
838  651
839  652
840  653
841  654
842  655
843  656
844  657
845  658
846  659
847  660
848  661
849  662
850  663
851  664
852  665
853  666
854  667
855  668
856  669
857  670
858  671
859  672
860  673
861  674
862  675
863  676
864  677
865  678
866  679
867  680
868  681
869  682
870  683
871  684
872  685
873  686
874  687
875  688
876  689
877  690
878  691
879  692
880  693
881  694
882  695
883  696
884  697
885  698
886  699
887  700
888  701
889  702
890  703
891  704
892  705
893  706
894  707
895  708
896  709
897  710
898  711
899  712
900  713
901  714
902  715
903  716
904  717
905  718
906  719
907  720
908  721
909  722
910  723
911  724
912  725
913  726
914  727
915  728
916  729
917  730
918  731
919  732
920  733
921  734
922  735
923  736
924  737
925  738
926  739
927  740
928  741
929  742
930  743
931  744
932  745
933  746
934  747
935  748
936  749
937  750
938  751
939  752
940  753
941  754
942  755
943  756
944  757
945  758
946  759
947  760
948  761
949  762
950  763
951  764
952  765
953  766
954  767
955  768
956  769
957  770
958  771
959  772
960  773
961  774
962  775
963  776
964  777
965  778
966  779
967  780
968  781
969  782
970  783
971  784
972  785
973  786
974  787
975  788
976  789
977  790
978  791
979  792
980  793
981  794
982  795
983  796
984  797
985  798
986  799
987  800
988  801
989  802
990  803
991  804
992  805
993  806
994  807
995  808
996  809
997  810
998  811
999  812
1000 813

```

Line	File	Code
351	2	/*
352	2	The parent has no need for the fd
353	2	so to close it.
354	2	* the fd used to read *from the parent.
355	2	* If these are not close ERROR and STOPPE
356	2	* may be misused by the writer when the
357	2	* intended reader dies.
358	2	*/
359	2	(void)close(cmd_pipe.toRED());
360	2	(void)close(cmd_pipe.toRED());
361	2	(void)close(cmd_pipe.fromRED());
362	2	(void)close(pipe.fromRED());
363	2	*/
364	2	/*
365	2	JOC does want to save the other fds
366	2	*/
367	2	xp->xp_fd_from_x = cmd_pipe.from(RED);
368	2	xp->xp_fd_to_x = cmd_pipe.to(RED);
369	2	xp->xp_fd_bulk_to_x = bulk_pipe.to(RED);
370	2	xp->xp_fd_pipe_from_x = pipe_pipe.from(RED);
371	2	*/
372	2	/* In debugmode, exec the separate-process
373	2	version of the augepsoc (grace issue)
374	2	*/
375	2	if (debugmode)
376	2	{
377	2	augepsocexec(xp->xp_fd_to_x, 'X', 0, "");
378	2	}
379	2	} //
380	2	define PING_TEST_STR "abc"
381	2	define PING_TEST_STR_SIZE 4 /* include the '\0' */
382	2	fd = xp->xp_fd_to_x;
383	2	augepsocexec(fd, 'P', PING_TEST_STR_SIZE, PING_TEST_STR);
384	2	fd = xp->xp_fd_from_x;
385	2	ping_status = augepsocexec(fd, 'P', 0, &resultbuf);
386	2	if (((! ping_status)
387	2	(NUL == resultbuf)
388	2	(strcmp(resultbuf, PING_TEST_STR) != 0))
389	2	{
390	2	fprintf(stderr, "Can't ping augepsoc, NUL");
391	2	fprintf(stderr, "Can't ping augepsoc, NUL");
392	2	return(AB_RECOVER, AUGEPROC_DIED);
393	2	}
394	2	free(resultbuf);
395	2	return(OK_SUCCESS);
396	1	*/
397	1	define PING_TEST_STR
398	1	define PING_TEST_STR_SIZE
399	1	define RED
400	1	define MAX_FD
401	1	/* end of setup_aux_process() */

Mon Oct 13 16:50:57 2008
SetupAUXProcess
Page 43 of 134

Mon Oct 13 16:00:37 2008	start_cpioexec	Page 44 of 134
315	defining MAX_SUBMIT_FIELD 2048	
316	*****	
317	start_cpioexec()	
318	*****	
319	* This function initializes a work item restore. It first determines	
320	* the size of the restore command to send to auxproc, malloc's the	
321	* memory and creates the restore command and sends it to auxproc.	
322	* Auxproc will start the cmd (if necessary) and start xlogproc.	
323	* Auxproc will send the initialization reply which is read by this	
324	* function. The results are for initialization.	
325		
326		
327	Args:	
328	ctx -- Recover Context struct (limited use)	
329	(i) xp -- auxproc status pointer.	
330	(i) submitObjCtxt -- Identifies what and how to run restores.	
331	(i) submitElemId -- Identifies what and how to run witem	restores.
332		
333	* (i) auxprocnum -- auxproc number may become obsolete.	
334	* (i) result -- the results of the restore.	
335	* (i) errors -- Of the auxproc restore initialization.	
336	* (i) errors2 -- Of work item initialization failures.	
337		
338	* Returns:	
339	* xlogproc's pid for success, -1 for failure.	
340		
341		
342	NOTES:	
343	ctx -- The recover context structure is carefully used below	
344	to store global values. It should be used only to get the global values	
345	like the xlogproc executable name and the config structure.	
346		
347	* Submit Object should be used to determine user id.	
348	* admin privileges, and other values that would not vary	
349	* for a potentially multi work item restore.	
350		
351	* Submit Element should be used to determine anything that	
352	* could be potentially unique for a work item restore.	
353		
354	* remoteId is the command line for the remote command.	
355	*****	
356	int	
357	start_cpioexec(struct recover_context *ctx,	
358	int xp,	
359	int submitObjCtxt,	
360	int submitElemId,	
361	char *auxprocnum,	
362	char *remoteId[4],	
363	rmt_tkno_tinfo ptkno_tinfo,	
364	rmt_tkno_tinfo *results,	
365	char *errors,	
366	char *errors2)	
367	{	
368	char *auxproc_datapath;	
369	struct data_len = 0;	
370	char *p;	
371	int i;	
372	int result;	
373		
374	/* For Initialization results */	
375	int i;	
376	int i2;	
377	rmt_tkno_tinfo *pctx;	
378	rmt_tkno_tinfo ptkno_tinfo;	
379	char *resultbuffer;	
380	char *resultbuffer2 = NULL;	
381	char *errors = "";	
382		
383	struct mark_summary submit_t_summary;	
384		
385		
386		
387		
388		
389		
390		
391		
392		
393		
394		
395		
396		
397		
398		
399		
400		
401		
402		
403		
404		
405		
406		
407		
408		
409		
410		
411		
412		
413		
414		
415		
416		
417		
418		
419		
420		
421		
422		
423		
424		
425		
426		
427		
428		
429		
430		
431		
432		
433		
434		
435		
436		
437		
438		
439		
440		
441		
442		
443		
444		
445		
446		
447		
448		
449		
450		
451		
452		
453		
454		
455		
456		
457		
458		
459		
460		
461		
462		
463		
464		
465		
466		
467		
468		
469		
470		
471		
472		
473		</

504.1	*/	
506.1	data_len += strlen(total_bytes_flag) + 1; /* xcplogon argv[5] */	
507.1	data_len += strlen(total_bytes_string) + 1; /* xcplogon argv[6] */	
509.1	/*	
510.1	* add size for db API socket info	
511.1	*/	
513.1	data_len += sizeof (int); /* socket port # */	
514.1	data_len += strlen(temp_socket_host) + 1;	
516.1	/*	
517.1	* locate work item in config info & get length of filetemp	
518.1	* changed break in inner loop to 'goto' to resume with	
519.1	* found item.	
520.1	*/	
522.1	for (pi = NULL, pj = rcx->rc.config->pproduct /* OK */;	
523.1	NULL != pj;	
524.1	pj = pj->next;	
525.1	{	
526.1	for (pi = pj->product; NULL != pi; pi = pi->next)	
527.1	{	
528.1	if (0 == strcmp(pi->name, temp_workitem.name))	
529.1	{	
530.1	goto stopsearch; /* \$\$\$ exit both loops */	
531.1	}	
532.1	}	
533.1	}	
534.1	}	
535.1	stopsearch:	
536.1	if (pi != NULL)	
537.1	{	
538.1	data_len += strlen(pi->list)*1;	
539.1	}	
540.1	else	
541.1	{	
542.1	data_len += 1;	
543.1	}	
545.1	if (NULL != pi && DEFAULT_DB_BUFFER_SIZE !=	
546.1	pi->recover_server_buffer_size)	
547.1	{	
548.1	sprintf(buffer, buffer, "-R%d", pi->recover_server_buffer_size);	
549.1	buffer_size = buffer_size + 1;	
550.1	data_len += strlen(buffer) + 1;	
551.1	xcplogon_argv++; /* one more arg to xcplogon */	
553.1	}	
555.1	data_len += strlen(temp_workitem.name) + 1;	
556.1	/*	
557.1	* allocate memory to hold all this junk we need to	
558.1	* shove towards our auxiliary process.	
559.1	*/	
560.1	auxproc_databuf = sm_malloc((unsigned) data_len);	
562.1	/*	
563.1	* fill in the junk. First, the rcmd info for the csh part.	
564.1	*/	
565.1	P = auxproc_databuf;	
566.1		

567.1	for (i = 0; i < 4; i++)	
568.1	{	
569.1	(void) strcpy(P, rcmdinfo[i]);	
570.1	P += strlen(P)+1;	
571.1	}	
573.1	/*	
574.1	* The human username which the remote should operate as	
575.1	*/	
577.1	(void) strcpy(P, temp_effective_username);	
578.1	P += strlen(P)+1;	
580.1	/*	
581.1	* The xcplogon cmd-fd info, which tells auxproc which	
582.1	* arg has the sprintf format string to insert the	
583.1	* actual output fd number. Remember that the "csh" arg	
584.1	* is the first arg, and the "rcmd" arg is the second	
585.1	* below arg is sent to auxproc to tell auxproc which	
586.1	* argument to update the format string with the outputfd.	
587.1	*/	
589.1	i = 3;	
590.1	/* for argv[3] -- See outputfd_fmt below. */	
591.1		
593.1	memory(P, &i, sizeof (int));	
594.1	P += sizeof (int);	
596.1	/*	
597.1	* the flags, which are always zero currently	
598.1	*/	
599.1	i = 0;	
600.1	memory(P, &i, sizeof (int));	
601.1	P += sizeof (int);	
603.1	/*	
604.1	* The "xcplogon command" that we will run locally.	
605.1	*/	
607.1	(void) strcpy(P, rcx -> rc_cpiogen_executable);	
608.1	P += strlen(P)+1;	
610.1	/*	
611.1	* The arg for the "xcplogon command", and its argv vector.	
612.1	*/	
614.1	memory(P, &xcplogon_argv, sizeof (int));	
615.1	P += sizeof (int);	
617.1	/*	
618.1	* The argv vector, which is argv[0], the outputfd	
619.1	* thing, and the progress report mode flag.	
620.1	*/	
622.1	(void) strcpy(P, xcplogon_argv[0]);	
623.1	P += strlen(P)+1;	
625.1	(void) strcpy(P, submit_file_flag); /* xcplogon argv[1] */	
626.1	P += strlen(submit_file_flag) + 1;	
628.1	(void) strcpy(P, temp_submit_file); /* xcplogon argv[2] */	
629.1	P += strlen(temp_submit_file) + 1;	
631.1	(void) strcpy(P, outputfd_fmt); /* xcplogon argv[3] */	
632.1	P += strlen(outputfd_fmt) + 1;	

```

614 1 (void)strcpy(p, progress_report_flag); /* xcploggen argv(4) */
615 1 p += strlen(progress_report_flag) + 1;
616 1
617 1 {
618 1     if (bufsizep != NULL)
619 1         p += strlen(bufsizep) + 1;
620 1     }
621 1
622 1 /*
623 1  * Follow this with the total bytes flag and value.
624 1  */
625 1
626 1 strcpy(p, total_bytes_flag); /* xcploggen argv(7)? */
627 1 p += strlen(total_bytes_flag) + 1;
628 1 strcpy(p, total_bytes_string); /* xcploggen argv(7)? */
629 1 p += strlen(total_bytes_string) + 1;
630 1
631 1 /*
632 1  * socket info for db API
633 1  */
634 1
635 1 (void)memcpy(p, (char *)&temp_socket_port, sizeof (int));
636 1 p += sizeof (int);
637 1
638 1 /*
639 1  * socket host name for db API
640 1  */
641 1
642 1 (void)strcpy(p, temp_socket_host);
643 1 p += strlen(p)+1;
644 1
645 1 if (NULL != p1) /* send filesizep */
646 1 {
647 1     (void)strcpy(p, p1->isac);
648 1     p += strlen(p)+1;
649 1 }
650 1 else
651 1 {
652 1     *p++ = 0;
653 1 }
654 1
655 1 /*
656 1  * workitem name
657 1  */
658 1
659 1 (void)strcpy(p, temp_workitem_name);
660 1 p += strlen(p)+1;
661 1
662 1 /*
663 1  * assert that our arithmetic above was done correctly
664 1  */
665 1
666 1 if ((size_t)(p - auxproc_databuf) != data_len)
667 1 {
668 1     the_log_stats(0, "assertion failed: cmd size mismatch");
669 1     return -1;
670 1 }
671 1
672 1 /*
673 1  * Send the restore command to auxproc. Auxproc will start
674 1  * The remote command (if necessary) and xcploggen.
675 1  */
676 1
677 1 auxprocpacket[0] = xp_fd.to_x;

```

```

699 1
700 1 /*
701 1  * Obtain the fork status
702 1  */
703 1
704 1 resultd = xp->xp_fd.from_x;
705 1
706 1 i = auxresults[resultd, '0', 0, &resultsbuflp];
707 1
708 1 {
709 1     if ((i < 0)
710 1         || the_log_stats(0,
711 1             "Error while starting auxproc for work item %s",
712 1             temp_workitem_name);
713 1         null_free(resultsbuflp);
714 1         return -1;
715 1     }
716 1
717 1 /* This memory management is crap */
718 1
719 1 pktdp = &pktdp_buf;
720 1 memcpy(pktdp, resultsbuflp, sizeof(pktdp_buf));
721 1 memory(resultd, resultsbuflp, sizeof(pktdp_buf));
722 1
723 1 if ((pktdp->magic) > 0)
724 1 {
725 1     resultd = resultsbuflp + sizeof "pktdp";
726 1     *err_str = &pktdp_buf;
727 1 }
728 1 else
729 1 {
730 1     resultd = "";
731 1     *err_str = &pktdp_buf;
732 1 }
733 1
734 1 /*
735 1  * If the fork failed, the cpiogen start fails
736 1  */
737 1
738 1 if (0 != pktdp->failcode)
739 1 {
740 1     int junk;
741 1
742 1     if (strlen(resultd) > (size_t)0)
743 1     {
744 1         the_log_stats(0,
745 1             "Error while starting auxproc, error %s",
746 1             "For work item '%s'", &resultd,
747 1             temp_workitem_name);
748 1     }
749 1     free(resultsbuflp);
750 1
751 1 /*
752 1  * collect the useless 'r' reply packet
753 1  */
754 1
755 1 resultbuflp = (char *)&junk;
756 1 (void)auxresults[resultd, 'r', sizeof (int), &resultbuflp];
757 1 return -1;
758 1 }
759 1
760 1

```

```

762 1      /*
763 1      * caller assumes responsibility for (eventually)
764 1      * collected exit status of remote and local programs.
765 1      */
766 1
767 1      pid = pkcOp->pid;
768 1      free (resultBufptr);
769 1
770 1      return pid;
771 1      /* end of start_cpioغن() */
772 1  }

```

```

773 1      /*
774 1      * 1) Remove rcx references.
775 1      */
776 1      static char *
777 1      make_remote_cpioغن, cmd(struct recover_context *rcx,
778 1                          int SubMITObjectID,
779 1                          int SubMITLEndID)
780 1      {
781 1          char *rcmpath = generate_rcmpath(SubMITObjectID,
782 1                                          SubMITLEndID);
783 1
784 1          char *minus_C = "-C";
785 1          char *minus_C_arg = "-C";
786 1          char *rclobber = "-C";
787 1          char *rclobber_arg = "-C";
788 1          char *rclobber_arg2 = "-C";
789 1          char *rclobber_arg3 = "-C";
790 1          char *rclobber_arg4 = "-C";
791 1          char *rclobber_arg5 = "-C";
792 1          char *rclobber_arg6 = "-C";
793 1          char *rclobber_arg7 = "-C";
794 1          char *rclobber_arg8 = "-C";
795 1          char *rclobber_arg9 = "-C";
796 1          char *rclobber_arg10 = "-C";
797 1          char *rclobber_arg11 = "-C";
798 1          char *rclobber_arg12 = "-C";
799 1          char *rclobber_arg13 = "-C";
800 1          char *rclobber_arg14 = "-C";
801 1          char *rclobber_arg15 = "-C";
802 1          char *rclobber_arg16 = "-C";
803 1          char *rclobber_arg17 = "-C";
804 1          char *rclobber_arg18 = "-C";
805 1          char *rclobber_arg19 = "-C";
806 1          char *rclobber_arg20 = "-C";
807 1          char *rclobber_arg21 = "-C";
808 1          char *rclobber_arg22 = "-C";
809 1          char *rclobber_arg23 = "-C";
810 1          char *rclobber_arg24 = "-C";
811 1          char *rclobber_arg25 = "-C";
812 1          char *rclobber_arg26 = "-C";
813 1          char *rclobber_arg27 = "-C";
814 1          char *rclobber_arg28 = "-C";
815 1          char *rclobber_arg29 = "-C";
816 1          char *rclobber_arg30 = "-C";
817 1          char *rclobber_arg31 = "-C";
818 1          char *rclobber_arg32 = "-C";
819 1          char *rclobber_arg33 = "-C";
820 1          char *rclobber_arg34 = "-C";
821 1          char *rclobber_arg35 = "-C";
822 1          char *rclobber_arg36 = "-C";
823 1          char *rclobber_arg37 = "-C";
824 1          char *rclobber_arg38 = "-C";
825 1          char *rclobber_arg39 = "-C";
826 1          char *rclobber_arg40 = "-C";
827 1          char *rclobber_arg41 = "-C";
828 1          char *rclobber_arg42 = "-C";
829 1          char *rclobber_arg43 = "-C";
830 1          char *rclobber_arg44 = "-C";
831 1          char *rclobber_arg45 = "-C";
832 1          char *rclobber_arg46 = "-C";
833 1          char *rclobber_arg47 = "-C";
834 1          char *rclobber_arg48 = "-C";
835 1          char *rclobber_arg49 = "-C";
836 1          char *rclobber_arg50 = "-C";
837 1          char *rclobber_arg51 = "-C";
838 1          char *rclobber_arg52 = "-C";
839 1          char *rclobber_arg53 = "-C";
840 1          char *rclobber_arg54 = "-C";
841 1          char *rclobber_arg55 = "-C";
842 1          char *rclobber_arg56 = "-C";
843 1          char *rclobber_arg57 = "-C";
844 1          char *rclobber_arg58 = "-C";
845 1          char *rclobber_arg59 = "-C";
846 1          char *rclobber_arg60 = "-C";
847 1          char *rclobber_arg61 = "-C";
848 1          char *rclobber_arg62 = "-C";
849 1          char *rclobber_arg63 = "-C";
850 1          char *rclobber_arg64 = "-C";
851 1          char *rclobber_arg65 = "-C";
852 1          char *rclobber_arg66 = "-C";
853 1          char *rclobber_arg67 = "-C";
854 1          char *rclobber_arg68 = "-C";
855 1          char *rclobber_arg69 = "-C";
856 1          char *rclobber_arg70 = "-C";
857 1          char *rclobber_arg71 = "-C";
858 1          char *rclobber_arg72 = "-C";
859 1          char *rclobber_arg73 = "-C";
860 1          char *rclobber_arg74 = "-C";
861 1          char *rclobber_arg75 = "-C";
862 1          char *rclobber_arg76 = "-C";
863 1          char *rclobber_arg77 = "-C";
864 1          char *rclobber_arg78 = "-C";
865 1          char *rclobber_arg79 = "-C";
866 1          char *rclobber_arg80 = "-C";
867 1          char *rclobber_arg81 = "-C";
868 1          char *rclobber_arg82 = "-C";
869 1          char *rclobber_arg83 = "-C";
870 1          char *rclobber_arg84 = "-C";
871 1          char *rclobber_arg85 = "-C";
872 1          char *rclobber_arg86 = "-C";
873 1          char *rclobber_arg87 = "-C";
874 1          char *rclobber_arg88 = "-C";
875 1          char *rclobber_arg89 = "-C";
876 1          char *rclobber_arg90 = "-C";
877 1          char *rclobber_arg91 = "-C";
878 1          char *rclobber_arg92 = "-C";
879 1          char *rclobber_arg93 = "-C";
880 1          char *rclobber_arg94 = "-C";
881 1          char *rclobber_arg95 = "-C";
882 1          char *rclobber_arg96 = "-C";
883 1          char *rclobber_arg97 = "-C";
884 1          char *rclobber_arg98 = "-C";
885 1          char *rclobber_arg99 = "-C";
886 1          char *rclobber_arg100 = "-C";
887 1          char *rclobber_arg101 = "-C";
888 1          char *rclobber_arg102 = "-C";
889 1          char *rclobber_arg103 = "-C";
890 1          char *rclobber_arg104 = "-C";
891 1          char *rclobber_arg105 = "-C";
892 1          char *rclobber_arg106 = "-C";
893 1          char *rclobber_arg107 = "-C";
894 1          char *rclobber_arg108 = "-C";
895 1          char *rclobber_arg109 = "-C";
896 1          char *rclobber_arg110 = "-C";
897 1          char *rclobber_arg111 = "-C";
898 1          char *rclobber_arg112 = "-C";
899 1          char *rclobber_arg113 = "-C";
900 1          char *rclobber_arg114 = "-C";
901 1          char *rclobber_arg115 = "-C";
902 1          char *rclobber_arg116 = "-C";
903 1          char *rclobber_arg117 = "-C";
904 1          char *rclobber_arg118 = "-C";
905 1          char *rclobber_arg119 = "-C";
906 1          char *rclobber_arg120 = "-C";
907 1          char *rclobber_arg121 = "-C";
908 1          char *rclobber_arg122 = "-C";
909 1          char *rclobber_arg123 = "-C";
910 1          char *rclobber_arg124 = "-C";
911 1          char *rclobber_arg125 = "-C";
912 1          char *rclobber_arg126 = "-C";
913 1          char *rclobber_arg127 = "-C";
914 1          char *rclobber_arg128 = "-C";
915 1          char *rclobber_arg129 = "-C";
916 1          char *rclobber_arg130 = "-C";
917 1          char *rclobber_arg131 = "-C";
918 1          char *rclobber_arg132 = "-C";
919 1          char *rclobber_arg133 = "-C";
920 1          char *rclobber_arg134 = "-C";
921 1          char *rclobber_arg135 = "-C";
922 1          char *rclobber_arg136 = "-C";
923 1          char *rclobber_arg137 = "-C";
924 1          char *rclobber_arg138 = "-C";
925 1          char *rclobber_arg139 = "-C";
926 1          char *rclobber_arg140 = "-C";
927 1          char *rclobber_arg141 = "-C";
928 1          char *rclobber_arg142 = "-C";
929 1          char *rclobber_arg143 = "-C";
930 1          char *rclobber_arg144 = "-C";
931 1          char *rclobber_arg145 = "-C";
932 1          char *rclobber_arg146 = "-C";
933 1          char *rclobber_arg147 = "-C";
934 1          char *rclobber_arg148 = "-C";
935 1          char *rclobber_arg149 = "-C";
936 1          char *rclobber_arg150 = "-C";
937 1          char *rclobber_arg151 = "-C";
938 1          char *rclobber_arg152 = "-C";
939 1          char *rclobber_arg153 = "-C";
940 1          char *rclobber_arg154 = "-C";
941 1          char *rclobber_arg155 = "-C";
942 1          char *rclobber_arg156 = "-C";
943 1          char *rclobber_arg157 = "-C";
944 1          char *rclobber_arg158 = "-C";
945 1          char *rclobber_arg159 = "-C";
946 1          char *rclobber_arg160 = "-C";
947 1          char *rclobber_arg161 = "-C";
948 1          char *rclobber_arg162 = "-C";
949 1          char *rclobber_arg163 = "-C";
950 1          char *rclobber_arg164 = "-C";
951 1          char *rclobber_arg165 = "-C";
952 1          char *rclobber_arg166 = "-C";
953 1          char *rclobber_arg167 = "-C";
954 1          char *rclobber_arg168 = "-C";
955 1          char *rclobber_arg169 = "-C";
956 1          char *rclobber_arg170 = "-C";
957 1          char *rclobber_arg171 = "-C";
958 1          char *rclobber_arg172 = "-C";
959 1          char *rclobber_arg173 = "-C";
960 1          char *rclobber_arg174 = "-C";
961 1          char *rclobber_arg175 = "-C";
962 1          char *rclobber_arg176 = "-C";
963 1          char *rclobber_arg177 = "-C";
964 1          char *rclobber_arg178 = "-C";
965 1          char *rclobber_arg179 = "-C";
966 1          char *rclobber_arg180 = "-C";
967 1          char *rclobber_arg181 = "-C";
968 1          char *rclobber_arg182 = "-C";
969 1          char *rclobber_arg183 = "-C";
970 1          char *rclobber_arg184 = "-C";
971 1          char *rclobber_arg185 = "-C";
972 1          char *rclobber_arg186 = "-C";
973 1          char *rclobber_arg187 = "-C";
974 1          char *rclobber_arg188 = "-C";
975 1          char *rclobber_arg189 = "-C";
976 1          char *rclobber_arg190 = "-C";
977 1          char *rclobber_arg191 = "-C";
978 1          char *rclobber_arg192 = "-C";
979 1          char *rclobber_arg193 = "-C";
980 1          char *rclobber_arg194 = "-C";
981 1          char *rclobber_arg195 = "-C";
982 1          char *rclobber_arg196 = "-C";
983 1          char *rclobber_arg197 = "-C";
984 1          char *rclobber_arg198 = "-C";
985 1          char *rclobber_arg199 = "-C";
986 1          char *rclobber_arg200 = "-C";
987 1          char *rclobber_arg201 = "-C";
988 1          char *rclobber_arg202 = "-C";
989 1          char *rclobber_arg203 = "-C";
990 1          char *rclobber_arg204 = "-C";
991 1          char *rclobber_arg205 = "-C";
992 1          char *rclobber_arg206 = "-C";
993 1          char *rclobber_arg207 = "-C";
994 1          char *rclobber_arg208 = "-C";
995 1          char *rclobber_arg209 = "-C";
996 1          char *rclobber_arg210 = "-C";
997 1          char *rclobber_arg211 = "-C";
998 1          char *rclobber_arg212 = "-C";
999 1          char *rclobber_arg213 = "-C";
1000 1         char *rclobber_arg214 = "-C";

```

```

832 1      NULL, NULL,
840 1      rcx->rc.config, /* OK */
841 1
842 1      if (work_item == NULL)
843 1          RBC_PIND_MORTTEN_NO_OPTIONS);
844 1      {
845 1          /* DEPRECATED: work_item->recover_client_bufferize)
846 1          {
847 1              sprintf(buffer, "%-A-b-%d",
848 1                  work_item->recover_client_bufferize);
849 1              bufferize = bufferize;
850 1          }
851 1          switch (temp_overwrite_policy)
852 1          {
853 1              case RC_OVERPOL_NO_CLOBBER:
854 1                  noclobber = "-A-overwrite";
855 1                  break;
856 1              case RC_OVERPOL_NEM_CLOBBER:
857 1                  noclobber = "-A-overwrite";
858 1                  break;
859 1              default:
860 1                  break; /* do something better here */
861 1          }
862 1          if (debugmode)
863 1          {
864 1              static char debugarg[100];
865 1              {
866 1                  (void) sprintf(debugarg, "-x /tmp/RBdebugd
867 1                  ", getpid());
868 1                  debugarg[flag] = debugarg;
869 1              }
870 1              len = strlen(debugarg) +
871 1                  strlen(debugarg) +
872 1                  strlen(debugarg) +
873 1                  strlen(debugarg) +
874 1                  strlen(debugarg) +
875 1                  strlen(debugarg) +
876 1                  strlen(debugarg) +
877 1                  strlen(debugarg) +
878 1                  strlen(debugarg) +
879 1                  strlen(debugarg) +
880 1                  strlen(debugarg) +
881 1                  strlen(debugarg) +
882 1                  strlen(debugarg) +
883 1                  strlen(debugarg) +
884 1                  strlen(debugarg) +
885 1                  strlen(debugarg) +
886 1                  strlen(debugarg) +
887 1                  strlen(debugarg) +
888 1                  strlen(debugarg) +
889 1                  strlen(debugarg) +
890 1                  strlen(debugarg) +
891 1                  strlen(debugarg) +
892 1                  strlen(debugarg) +
893 1                  strlen(debugarg) +
894 1                  strlen(debugarg) +
895 1                  strlen(debugarg) +
896 1                  strlen(debugarg) +
897 1                  strlen(debugarg) +
898 1                  strlen(debugarg) +
899 1                  strlen(debugarg) +
900 1                  strlen(debugarg) +
901 1                  strlen(debugarg) +
902 1                  strlen(debugarg) +
903 1                  strlen(debugarg) +
904 1                  strlen(debugarg) +
905 1                  strlen(debugarg) +
906 1                  strlen(debugarg) +
907 1                  strlen(debugarg) +
908 1                  strlen(debugarg) +
909 1                  strlen(debugarg) +
910 1                  strlen(debugarg) +
911 1                  strlen(debugarg) +
912 1                  strlen(debugarg) +
913 1                  strlen(debugarg) +
914 1                  strlen(debugarg) +
915 1                  strlen(debugarg) +
916 1                  strlen(debugarg) +
917 1                  strlen(debugarg) +
918 1                  strlen(debugarg) +
919 1                  strlen(debugarg) +
920 1                  strlen(debugarg) +
921 1                  strlen(debugarg) +
922 1                  strlen(debugarg) +
923 1                  strlen(debugarg) +
924 1                  strlen(debugarg) +
925 1                  strlen(debugarg) +
926 1                  strlen(debugarg) +
927 1                  strlen(debugarg) +
928 1                  strlen(debugarg) +
929 1                  strlen(debugarg) +
930 1                  strlen(debugarg) +
931 1                  strlen(debugarg) +
932 1                  strlen(debugarg) +
933 1                  strlen(debugarg) +
934 1                  strlen(debugarg) +
935 1                  strlen(debugarg) +
936 1                  strlen(debugarg) +
937 1                  strlen(debugarg) +
938 1                  strlen(debugarg) +
939 1                  strlen(debugarg) +
940 1                  strlen(debugarg) +
941 1                  strlen(debugarg) +
942 1                  strlen(debugarg) +
943 1                  strlen(debugarg) +
944 1                  strlen(debugarg) +
945 1                  strlen(debugarg) +
946 1                  strlen(debugarg) +
947 1                  strlen(debugarg) +
948 1                  strlen(debugarg) +
949 1                  strlen(debugarg) +
950 1                  strlen(debugarg) +
951 1                  strlen(debugarg) +
952 1                  strlen(debugarg) +
953 1                  strlen(debugarg) +
954 1                  strlen(debugarg) +
955 1                  strlen(debugarg) +
956 1                  strlen(debugarg) +
957 1                  strlen(debugarg) +
958 1                  strlen(debugarg) +
959 1                  strlen(debugarg) +
960 1                  strlen(debugarg) +
961 1                  strlen(debugarg) +
962 1                  strlen(debugarg) +
963 1                  strlen(debugarg) +
964 1                  strlen(debugarg) +
965 1                  strlen(debugarg) +
966 1                  strlen(debugarg) +
967 1                  strlen(debugarg) +
968 1                  strlen(debugarg) +
969 1                  strlen(debugarg) +
970 1                  strlen(debugarg) +
971 1                  strlen(debugarg) +
972 1                  strlen(debugarg) +
973 1                  strlen(debugarg) +
974 1                  strlen(debugarg) +
975 1                  strlen(debugarg) +
976 1                  strlen(debugarg) +
977 1                  strlen(debugarg) +
978 1                  strlen(debugarg) +
979 1                  strlen(debugarg) +
980 1                  strlen(debugarg) +
981 1                  strlen(debugarg) +
982 1                  strlen(debugarg) +
983 1                  strlen(debugarg) +
984 1                  strlen(debugarg) +
985 1                  strlen(debugarg) +
986 1                  strlen(debugarg) +
987 1                  strlen(debugarg) +
988 1                  strlen(debugarg) +
989 1                  strlen(debugarg) +
990 1                  strlen(debugarg) +
991 1                  strlen(debugarg) +
992 1                  strlen(debugarg) +
993 1                  strlen(debugarg) +
994 1                  strlen(debugarg) +
995 1                  strlen(debugarg) +
996 1                  strlen(debugarg) +
997 1                  strlen(debugarg) +
998 1                  strlen(debugarg) +
999 1                  strlen(debugarg) +
1000 1                 strlen(debugarg) +

```

```

894 1      /*
895 1      * 1) Remove rc references.
896 1      * 2) Reservoir whether the normalise updates.
897 1      */
898 1      /*
899 1      * Construct the path name of the remote command that
900 1      * will be executed on the destination client.
901 1      * Return ptr to constructed string.
902 1      * NOTE: caller must copy if string is to be preserved.
903 1      */
904 1      static char *
905 1      generate_remote_cpipgen(int submit_obj_id,
906 1      int submit_obj_id)
907 1      {
908 1          static char *output = NULL;
909 1          size_t len_needed;
910 1          char *lph; /* pointer to %h */
911 1          char *p;
912 1          char *q;
913 1          char *normal_host;
914 1          char *temp_scriptname;
915 1          char *temp_client_hostname;
916 1          char *temp_client_hostname;
917 1          char *temp_client_hostname;
918 1          char *temp_client_hostname;
919 1          char *temp_client_hostname;
920 1          char *temp_client_hostname;
921 1          char *temp_client_hostname;
922 1          char *temp_client_hostname;
923 1          char *temp_client_hostname;
924 1          char *temp_client_hostname;
925 1          char *temp_client_hostname;
926 1          char *temp_client_hostname;
927 1          char *temp_client_hostname;
928 1          char *temp_client_hostname;
929 1          char *temp_client_hostname;
930 1          char *temp_client_hostname;
931 1          char *temp_client_hostname;
932 1          char *temp_client_hostname;
933 1          char *temp_client_hostname;
934 1          char *temp_client_hostname;
935 1          char *temp_client_hostname;
936 1          char *temp_client_hostname;
937 1          char *temp_client_hostname;
938 1          char *temp_client_hostname;
939 1          char *temp_client_hostname;
940 1          char *temp_client_hostname;
941 1          char *temp_client_hostname;
942 1          char *temp_client_hostname;
943 1          char *temp_client_hostname;
944 1          char *temp_client_hostname;
945 1          char *temp_client_hostname;
946 1          char *temp_client_hostname;
947 1          char *temp_client_hostname;
948 1          char *temp_client_hostname;
949 1          char *temp_client_hostname;
950 1          char *temp_client_hostname;
951 1          char *temp_client_hostname;
952 1          char *temp_client_hostname;
953 1          char *temp_client_hostname;
954 1          char *temp_client_hostname;
955 1          char *temp_client_hostname;
956 1          char *temp_client_hostname;
957 1          char *temp_client_hostname;
958 1          char *temp_client_hostname;
959 1          char *temp_client_hostname;
960 1          char *temp_client_hostname;
961 1          char *temp_client_hostname;
962 1          char *temp_client_hostname;
963 1          char *temp_client_hostname;
964 1          char *temp_client_hostname;
965 1          char *temp_client_hostname;
966 1          char *temp_client_hostname;
967 1          char *temp_client_hostname;
968 1          char *temp_client_hostname;
969 1          char *temp_client_hostname;
970 1          char *temp_client_hostname;
971 1          char *temp_client_hostname;
972 1          char *temp_client_hostname;
973 1          char *temp_client_hostname;
974 1          char *temp_client_hostname;
975 1          char *temp_client_hostname;
976 1          char *temp_client_hostname;
977 1          char *temp_client_hostname;
978 1          char *temp_client_hostname;
979 1          char *temp_client_hostname;
980 1          char *temp_client_hostname;
981 1          char *temp_client_hostname;
982 1          char *temp_client_hostname;
983 1          char *temp_client_hostname;
984 1          char *temp_client_hostname;
985 1          char *temp_client_hostname;
986 1          char *temp_client_hostname;
987 1          char *temp_client_hostname;
988 1          char *temp_client_hostname;
989 1          char *temp_client_hostname;
990 1          char *temp_client_hostname;
991 1          char *temp_client_hostname;
992 1          char *temp_client_hostname;
993 1          char *temp_client_hostname;
994 1          char *temp_client_hostname;
995 1          char *temp_client_hostname;
996 1          char *temp_client_hostname;
997 1          char *temp_client_hostname;
998 1          char *temp_client_hostname;
999 1          char *temp_client_hostname;
1000 1         char *temp_client_hostname;

```



```

989 2 {
990 1     return temp_scriptname;
991 1 }
992 1 /*
993 1  * here for a sh ... insert the destination client
994 1  * name into the rc client's scriptname. First
995 1  * compute how much storage will be needed to
996 1  * hold the result.
997 1  */
998 1 if ( (dcb_cen.itc.be.normalized(temp_client_hostname))
999 1     {
1000 1         rc = ap1_log_cen(SUB_CEN_NOMEM, NULL);
1001 1         the_log_stats(
1002 1             0, "Could not allocate memory generate_crandash!");
1003 1         /*
1004 1          * StartDonec()Back(RP_RL_RECOVER_FAILURES)*/
1005 1         return NULL;
1006 1     }
1007 1     nora_host = dcb_normalize(temp_client_hostname);
1008 1     len_needed = strlen(temp_scriptname) - 2 +
1009 1         /* -2: %s */
1010 1         /* +1: \0 */
1011 1         strlen(nora_host) + 1;
1012 1     if (mybuf == NULL || strlen(mybuf) < (size_t)len_needed-1)
1013 1     {
1014 1         if (mybuf != NULL)
1015 1             free(mybuf);
1016 1         if (mybuf = malloc((int)len_needed) == NULL)
1017 1         {
1018 1             rc = ap1_log_cen(SUB_CEN_NOMEM, NULL);
1019 1             the_log_stats(
1020 1                 0, "Could not allocate memory generate_crandash!");
1021 1             return NULL;
1022 1         }
1023 1     }
1024 1     q = mybuf;
1025 1     for (q = mybuf; p = temp_scriptname; p < pph; q++, p++)
1026 1     {
1027 1         *q = *p;
1028 1     }
1029 1     (void)strcpy(mybuf, nora_host);
1030 1     (void)strcat(mybuf, pph+2);
1031 1     return mybuf;
1032 1     /* end of generate_crandash() */
1033 1 }
1034 1
1035 1
1036 1
1037 1
1038 1
1039 1
1040 1
1041 1
1042 1
1043 1
1044 1
1045 1
1046 1
1047 1
1048 1
1049 1
1050 1
1051 1
1052 1
1053 1
1054 1
1055 1
1056 1
1057 1
1058 1
1059 1
1060 1
1061 1
1062 1
1063 1
1064 1
1065 1
1066 1
1067 1

```

```

1068 1 int
1069 1 StartWorkItemRecovery(struct recover_context *rcx,
1070 1     struct arboObject*
1071 1     struct arboObject*
1072 1     int SubmitItemID)
1073 1 {
1074 1     char
1075 1         *rcmdv[4]; /* 0 hostname, 1 louser, 2 remove, 3 cmd */
1076 1     int
1077 1         changed_priv = 0;
1078 1     int
1079 1         changed_effective_uid;
1080 1     int
1081 1         pldio_effective_uid;
1082 1     char *err_str_buffer = NULL;
1083 1     char temp_client_hostname(SHRTV_FIELD_MAX);
1084 1     char temp_client_hostname(SHRTV_FIELD_MAX);
1085 1     long temp_effective_uid;
1086 1     boolean_t temp_src_sysadmin;
1087 1     int GetSrcStatus = 0;
1088 1     int GetDstStatus = 0;
1089 1     if ( ( GetSrcClientName(SubmitObjectID, SubmitItemID,
1090 1         temp_client_hostname,
1091 1         SHRTV_FIELD_MAX,
1092 1         GetDstStatus) != 0 )
1093 1         {
1094 1             the_log_stats(0, "Unable to get client destination name.");
1095 1             return -1;
1096 1         }
1097 1     if ( GetSrcClientUserName(SubmitObjectID, SubmitItemID,
1098 1         SubmitItemID,
1099 1         SHRTV_FIELD_MAX,
1100 1         GetDstStatus) != 0 )
1101 1     {
1102 1         the_log_stats(0, "Unable to get client user name.");
1103 1         return -1;
1104 1     }
1105 1     temp_src_sysadmin = GetSrcSourceSysAdmin(SubmitObjectID,
1106 1         GetSrcStatus);
1107 1     if ( 0 != GetSrcStatus )
1108 1     {
1109 1         the_log_stats(
1110 1             0, "Unable to get the source system admin privileges.");
1111 1         return -1;
1112 1     }
1113 1     if ( ( GetSrcEffectiveUID(SubmitObjectID,
1114 1         SubmitItemID,
1115 1         GetSrcStatus) != 0 )
1116 1         {
1117 1             the_log_stats(0, "Unable to get effective uid.");
1118 1             return -1;
1119 1         }
1120 1     if ( 0 != (temp_src_sysadmin || 0 != temp_effective_uid)
1121 1         {
1122 1             set_recovery_privileges(rcx, changed_priv);
1123 1         }
1124 1     }
1125 1 }
1126 1
1127 1
1128 1
1129 1
1130 1
1131 1
1132 1
1133 1
1134 1
1135 1
1136 1
1137 1
1138 1
1139 1
1140 1
1141 1
1142 1
1143 1
1144 1
1145 1
1146 1
1147 1
1148 1
1149 1
1150 1
1151 1
1152 1
1153 1
1154 1
1155 1
1156 1
1157 1
1158 1
1159 1
1160 1
1161 1
1162 1
1163 1
1164 1
1165 1
1166 1
1167 1
1168 1
1169 1
1170 1
1171 1
1172 1
1173 1
1174 1
1175 1
1176 1
1177 1
1178 1
1179 1
1180 1
1181 1
1182 1
1183 1
1184 1
1185 1
1186 1
1187 1
1188 1
1189 1
1190 1
1191 1
1192 1
1193 1
1194 1
1195 1
1196 1
1197 1
1198 1
1199 1
1200 1
1201 1
1202 1
1203 1
1204 1
1205 1
1206 1
1207 1
1208 1
1209 1
1210 1
1211 1
1212 1
1213 1
1214 1
1215 1
1216 1
1217 1
1218 1
1219 1
1220 1
1221 1
1222 1
1223 1
1224 1
1225 1
1226 1
1227 1
1228 1
1229 1
1230 1
1231 1
1232 1
1233 1
1234 1
1235 1
1236 1
1237 1
1238 1
1239 1
1240 1
1241 1
1242 1
1243 1
1244 1
1245 1
1246 1
1247 1
1248 1
1249 1
1250 1
1251 1
1252 1
1253 1
1254 1
1255 1
1256 1
1257 1
1258 1
1259 1
1260 1
1261 1
1262 1
1263 1
1264 1
1265 1
1266 1
1267 1
1268 1
1269 1
1270 1
1271 1
1272 1
1273 1
1274 1
1275 1
1276 1
1277 1
1278 1
1279 1
1280 1
1281 1
1282 1
1283 1
1284 1
1285 1
1286 1
1287 1
1288 1
1289 1
1290 1
1291 1
1292 1
1293 1
1294 1
1295 1
1296 1
1297 1
1298 1
1299 1
1300 1
1301 1
1302 1
1303 1
1304 1
1305 1
1306 1
1307 1
1308 1
1309 1
1310 1
1311 1
1312 1
1313 1
1314 1
1315 1
1316 1
1317 1
1318 1
1319 1
1320 1
1321 1
1322 1
1323 1
1324 1
1325 1
1326 1
1327 1
1328 1
1329 1
1330 1
1331 1
1332 1
1333 1
1334 1
1335 1
1336 1
1337 1
1338 1
1339 1
1340 1
1341 1
1342 1
1343 1
1344 1
1345 1
1346 1
1347 1
1348 1
1349 1
1350 1
1351 1
1352 1
1353 1
1354 1
1355 1
1356 1
1357 1
1358 1
1359 1
1360 1
1361 1
1362 1
1363 1
1364 1
1365 1
1366 1
1367 1
1368 1
1369 1
1370 1
1371 1
1372 1
1373 1
1374 1
1375 1
1376 1
1377 1
1378 1
1379 1
1380 1
1381 1
1382 1
1383 1
1384 1
1385 1
1386 1
1387 1
1388 1
1389 1
1390 1
1391 1
1392 1
1393 1
1394 1
1395 1
1396 1
1397 1
1398 1
1399 1
1400 1
1401 1
1402 1
1403 1
1404 1
1405 1
1406 1
1407 1
1408 1
1409 1
1410 1
1411 1
1412 1
1413 1
1414 1
1415 1
1416 1
1417 1
1418 1
1419 1
1420 1
1421 1
1422 1
1423 1
1424 1
1425 1
1426 1
1427 1
1428 1
1429 1
1430 1
1431 1
1432 1
1433 1
1434 1
1435 1
1436 1
1437 1
1438 1
1439 1
1440 1
1441 1
1442 1
1443 1
1444 1
1445 1
1446 1
1447 1
1448 1
1449 1
1450 1
1451 1
1452 1
1453 1
1454 1
1455 1
1456 1
1457 1
1458 1
1459 1
1460 1
1461 1
1462 1
1463 1
1464 1
1465 1
1466 1
1467 1
1468 1
1469 1
1470 1
1471 1
1472 1
1473 1
1474 1
1475 1
1476 1
1477 1
1478 1
1479 1
1480 1
1481 1
1482 1
1483 1
1484 1
1485 1
1486 1
1487 1
1488 1
1489 1
1490 1
1491 1
1492 1
1493 1
1494 1
1495 1
1496 1
1497 1
1498 1
1499 1
1500 1
1501 1
1502 1
1503 1
1504 1
1505 1
1506 1
1507 1
1508 1
1509 1
1510 1
1511 1
1512 1
1513 1
1514 1
1515 1
1516 1
1517 1
1518 1
1519 1
1520 1
1521 1
1522 1
1523 1
1524 1
1525 1
1526 1
1527 1
1528 1
1529 1
1530 1
1531 1
1532 1
1533 1
1534 1
1535 1
1536 1
1537 1
1538 1
1539 1
1540 1
1541 1
1542 1
1543 1
1544 1
1545 1
1546 1
1547 1
1548 1
1549 1
1550 1
1551 1
1552 1
1553 1
1554 1
1555 1
1556 1
1557 1
1558 1
1559 1
1560 1
1561 1
1562 1
1563 1
1564 1
1565 1
1566 1
1567 1
1568 1
1569 1
1570 1
1571 1
1572 1
1573 1
1574 1
1575 1
1576 1
1577 1
1578 1
1579 1
1580 1
1581 1
1582 1
1583 1
1584 1
1585 1
1586 1
1587 1
1588 1
1589 1
1590 1
1591 1
1592 1
1593 1
1594 1
1595 1
1596 1
1597 1
1598 1
1599 1
1600 1
1601 1
1602 1
1603 1
1604 1
1605 1
1606 1
1607 1
1608 1
1609 1
1610 1
1611 1
1612 1
1613 1
1614 1
1615 1
1616 1
1617 1
1618 1
1619 1
1620 1
1621 1
1622 1
1623 1
1624 1
1625 1
1626 1
1627 1
1628 1
1629 1
1630 1
1631 1
1632 1
1633 1
1634 1
1635 1
1636 1
1637 1
1638 1
1639 1
1640 1
1641 1
1642 1
1643 1
1644 1
1645 1
1646 1
1647 1
1648 1
1649 1
1650 1
1651 1
1652 1
1653 1
1654 1
1655 1
1656 1
1657 1
1658 1
1659 1
1660 1
1661 1
1662 1
1663 1
1664 1
1665 1
1666 1
1667 1
1668 1
1669 1
1670 1
1671 1
1672 1
1673 1
1674 1
1675 1
1676 1
1677 1
1678 1
1679 1
1680 1
1681 1
1682 1
1683 1
1684 1
1685 1
1686 1
1687 1
1688 1
1689 1
1690 1
1691 1
1692 1
1693 1
1694 1
1695 1
1696 1
1697 1
1698 1
1699 1
1700 1
1701 1
1702 1
1703 1
1704 1
1705 1
1706 1
1707 1
1708 1
1709 1
1710 1
1711 1
1712 1
1713 1
1714 1
1715 1
1716 1
1717 1
1718 1
1719 1
1720 1
1721 1
1722 1
1723 1
1724 1
1725 1
1726 1
1727 1
1728 1
1729 1
1730 1
1731 1
1732 1
1733 1
1734 1
1735 1
1736 1
1737 1
1738 1
1739 1
1740 1
1741 1
1742 1
1743 1
1744 1
1745 1
1746 1
1747 1
1748 1
1749 1
1750 1
1751 1
1752 1
1753 1
1754 1
1755 1
1756 1
1757 1
1758 1
1759 1
1760 1
1761 1
1762 1
1763 1
1764 1
1765 1
1766 1
1767 1
1768 1
1769 1
1770 1
1771 1
1772 1
1773 1
1774 1
1775 1
1776 1
1777 1
1778 1
1779 1
1780 1
1781 1
1782 1
1783 1
1784 1
1785 1
1786 1
1787 1
1788 1
1789 1
1790 1
1791 1
1792 1
1793 1
1794 1
1795 1
1796 1
1797 1
1798 1
1799 1
1800 1
1801 1
1802 1
1803 1
1804 1
1805 1
1806 1
1807 1
1808 1
1809 1
1810 1
1811 1
1812 1
1813 1
1814 1
1815 1
1816 1
1817 1
1818 1
1819 1
1820 1
1821 1
1822 1
1823 1
1824 1
1825 1
1826 1
1827 1
1828 1
1829 1
1830 1
1831 1
1832 1
1833 1
1834 1
1835 1
1836 1
1837 1
1838 1
1839 1
1840 1
1841 1
1842 1
1843 1
1844 1
1845 1
1846 1
1847 1
1848 1
1849 1
1850 1
1851 1
1852 1
1853 1
1854 1
1855 1
1856 1
1857 1
1858 1
1859 1
1860 1
1861 1
1862 1
1863 1
1864 1
1865 1
1866 1
1867 1
1868 1
1869 1
1870 1
1871 1
1872 1
1873 1
1874 1
1875 1
1876 1
1877 1
1878 1
1879 1
1880 1
1881 1
1882 1
1883 1
1884 1
1885 1
1886 1
1887 1
1888 1
1889 1
1890 1
1891 1
1892 1
1893 1
1894 1
1895 1
1896 1
1897 1
1898 1
1899 1
1900 1
1901 1
1902 1
1903 1
1904 1
1905 1
1906 1
1907 1
1908 1
1909 1
1910 1
1911 1
1912 1
1913 1
1914 1
1915 1
1916 1
1917 1
1918 1
1919 1
1920 1
1921 1
1922 1
1923 1
1924 1
1925 1
1926 1
1927 1
1928 1
1929 1
1930 1
1931 1
1932 1
1933 1
1934 1
1935 1
1936 1
1937 1
1938 1
1939 1
1940 1
1941 1
1942 1
1943 1
1944 1
1945 1
1946 1
1947 1
1948 1
1949 1
1950 1
1951 1
1952 1
1953 1
1954 1
1955 1
1956 1
1957 1
1958 1
1959 1
1960 1
1961 1
1962 1
1963 1
1964 1
1965 1
1966 1
1967 1
1968 1
1969 1
1970 1
1971 1
1972 1
1973 1
1974 1
1975 1
1976 1
1977 1
1978 1
1979 1
1980 1
1981 1
1982 1
1983 1
1984 1
1985 1
1986 1
1987 1
1988 1
1989 1
1990 1
1991 1
1992 1
1993 1
1994 1
1995 1
1996 1
1997 1
1998 1
1999 1
2000 1
2001 1
2002 1
2003 1
2004 1
2005 1
2006 1
2007 1
2008 1
2009 1
2010 1
2011 1
2012 1
2013 1
2014 1
2015 1
2016 1
2017 1
2018 1
2019 1
2020 1
2021 1
2022 1
2023 1
2024 1
2025 1
2026 1
2027 1
2028 1
2029 1
2030 1
2031 1
2032 1
2033 1
2034 1
2035 1
2036 1
2037 1
2038 1
2039 1
2040 1
2041 1
2042 1
2043 1
2044 1
2045 1
2046 1
2047 1
2048 1
2049 1
2050 1
2051 1
2052 1
2053 1
2054 1
2055 1
2056 1
2057 1
2058 1
2059 1
2060 1
2061 1
2062 1
2063 1
2064 1
2065 1
2066 1
2067 1
2068 1
2069 1
2070 1
2071 1
2072 1
2073 1
2074 1
2075 1
2076 1
2077 1
2078 1
2079 1
2080 1
2081 1
2082 1
2083 1
2084 1
2085 1
2086 1
2087 1
2088 1
2089 1
2090 1
2091 1
2092 1
2093 1
2094 1
2095 1
2096 1
2097 1
2098 1
2099 1
2100 1
2101 1
2102 1
2103 1
2104 1
2105 1
2106 1
2107 1
2108 1
2109 1
2110 1
2111 1
2112 1
2113 1
2114 1
2115 1
2116 1
2117 1
2118 1
2119 1
2120 1
2121 1
2122 1
2123 1
2124 1
2125 1
2126 1
2127 1
2128 1
2129 1
2130 1
2131 1
2132 1
2133 1
2134 1
2135 1
2136 1
2137 1
2138 1
2139 1
2140 1
2141 1
2142 1
2143 1
2144 1
2145 1
2146 1
2147 1
2148 1
2149 1
2150 1
2151 1
2152 1
2153 1
2154 1
2155 1
2156 1
2157 1
2158 1
2159 1
2160 1
2161 1
2162 1
2163 1
2164 1
2165 1
2166 1
2167 1
2168 1
2169 1
2170 1
2171 1
2172 1
2173 1
2174 1
2175 1
2176 1
2177 1
2178 1
2179 1
2180 1
2181 1
2182 1
2183 1
2184 1
2185 1
2186 1
2187 1
2188 1
2189 1
2190 1
2191 1
2192 1
2193 1
2194 1
2195 1
2196 1
2197 1
2198 1
2199 1
2200 1
2201 1
2202 1
2203 1
2204 1
2205 1
2206 1
2207 1
2208 1
2209 1
2210 1
2211 1
2212 1
2213 1
2214 1
2215 1
2216 1
2217 1
2218 1
2219 1
2220 1
2221 1
2222 1
2223 1
2224 1
2225 1
2226 1
2227 1
2228 1
2229 1
2230 1
2231 1
2232 1
2233 1
2234 1
2235 1
2236 1
2237 1
2238 1
2239 1
2240 1
2241 1
2242 1
2243 1
2244 1
2245 1
2246 1
2247 1
2248 1
2249 1
2250 1
2251 1
2252 1
2253 1
2254 1
2255 1
2256 1
2257 1
2258 1
2259 1
2260 1
2261 1
2262 1
2263 1
2264 1
2265 1
2266 1
2267 1
2268 1
2269 1
2270 1
2271 1
2272 1
2273 1
2274 1
2275 1
2276 1
2277 1
2278 1
2279 1
2280 1
2281 1
2282 1
2283 1
2284 1
2285 1
2286 1
2287 1
2288 1
2289 1
2290 1
2291 1
2292 1
2293 1
2294 1
2295 1
2296 1
2297 1
2298 1
2299 1
2300 1
2301 1
2302 1
2303 1
2304 1
2305 1
2306 1
2307 1
2308 1
2309 1
2310 1
2311 1
2312 1
2313 1
2314 1
2315 1
2316 1
2317 1
2318 1
2319 1
2320 1
2321 1
2322 1
2323 1
2324 1
2325 1
2326 1
2327 1
2328 1
2329 1
2330 1
2331 1
2332 1
2333 1
2334 1
2335 1
2336 1
2337 1
2338 1
2339 1
2340 1
2341 1
2342 1
2343 1
2344 1
2345 1
2346 1
2347 1
2348 1
2349 1
2350 1
2351 1
2352 1
2353 1
2354 1
2355 1
2356 1
2357 1
2358 1
2359 1
2360 1
2361 1
2362 1
2363 1
2364 1
2365 1
2366 1
2367 1
2368 1
2369 1
2370 1
2371 1
2372 1
2373 1
2374 1
2375 1
2376 1
2377 1
2378 1
2379 1
2380 1
2381 1
2382 1
2383 1
2384 1
2385 1
2386 1
2387 1
2388 1
2389 1
2390 1
2391 1
2392 1
2393 1
2394 1
2395 1
2396 1
2397 1
2398 1
2399 1
2400 1
2401 1
2402 1
2403 1
2404 1
2405 1
2406 1
2407 1
2408 1
2409 1
2410 1
2411 1
2412 1
2413 1
2414 1
2415 1
2416 1
2417 1
2418 1
2419 1
2420 1
2421 1
2422 1
2423 1
2424 1
2425 1
2426 1
2427 1
2428 1
2429 1
2430 1
2431 1
2432 1
2433 1
2434 1
2435 1
2436 1
2437 1
2438 1
2439 1
2440 1
2441 1
2442 1
2443 1
2444 1
2445 1
2446 1
2447 1
2448 1
2449 1
2450 1
2451 1
2452 1
2453 1
2454 1
2455 1
2456 1
2457 1
2458 1
2459 1
2460 1
2461 1
2462 1
2463 1
2464 1
2465 1
2466 1
2467 1
2468 1
2469 1
2470 1
2471 1
2472 1
2473 1
2474 1
2475 1
2476 1
2477 1
2478 1
2479 1
2480 1
2481 1
2482 1
2483 1
2484 1
2485 1
2486 1
2487 1
2488 1
2489 1
2490 1
2491 1
2492 1
2493 1
2494 1
2495 1
2496 1
2497 1
2498 1
2499 1
2500 1
2501 1
2502 1
2503 1
2504 1
2505 1
2506 1
2507 1
2508 1
2509 1
2510 1
2511 1
2512 1
2513 1
2514 1
2515 1
2516 1
2517 1
2518 1
2519 1
2520 1
2521 1
2522 1
2523 1
2524 1
2525 1
2526 1
2527 1
2528 1
2529 1
2530 1
2531 1
2532 1
2533 1
2534 1
2535 1
2536 1
2537 1
2538 1
2539 1
2540 1
2541 1
2542 1
2543 1
2544 1
2545 1
2546 1
2547 1
2548 1
2549 1
2550 1
2551 1
2552 1
2553 1
2554 1
2555 1
2556 1
2557 1
2558 1
2559 1
2560 1
2561 1
2562 1
2563 1
2564 1
2565 1
2566 1
2567 1
2568 1
2569 1
2570 1
2571 1
2572 1
2573 1
2574 1
2575 1
2576 1
2577 1
2578 1
2579 1
2580 1
2581 1
258
```

```

1074 1 rcmdv[0] = temp_client_hostname;
1075 1 rcmdv[1] = temp_client_username;
1076 1 rcmdv[2] = temp_client_username;
1077 1 rcmdv[3] = make_remote_cpilogen_cmd(rcx,
1078 1 SubmitObjectID,
1079 1 SubmitItemID);
1080 1
1081 1 if (NULL == rcmdv[3])
1082 1 {
1083 1     return -1;
1084 1 }
1085 1 xcpilogen_pid = start_cpilogen(rcx, XO,
1086 1 SubmitObjectID,
1087 1 SubmitItemID,
1088 1 0,
1089 1 rcmdv,
1090 1 socktdp_buffer,
1091 1 socktdp_buffer);
1092 1
1093 1 if (changed_priv)
1094 1 {
1095 1     reset_recovery_privileges(rcx, changed_priv);
1096 1 }
1097 1 return xcpilogen_pid;
1098 1 } /* StartWorkItemRecovery() */

```

```

1100 1 /*
1101 1 * Check to see if the user has root access to the
1102 1 * destination client. If so, give him/her the root
1103 1 * privileges on the recovery.
1104 1 */
1105 1 static void
1106 1 sel_recovery_privileges(struct recover_context *rcx,
1107 1 int *changed)
1108 1 {
1109 1     int root_access;
1110 1     eperrno errnum;
1111 1
1112 1     *changed = 0;
1113 1     (void)rbcc_cancelrecover(rcx->rc.config, rcx->rc.client_hostname,
1114 1 rcx->rc.channel_username, &root_access,
1115 1 &errnum);
1116 1
1117 1 if (root_access)
1118 1 {
1119 1     if (debugmode)
1120 1     {
1121 1         rbe_log_status(
1122 1             0, "the user is a sys admin for the dest client");
1123 1     }
1124 1     rcx->rc.recovery_flags |= RC_RECPFLAG_DEST_SYSADMIN;
1125 1     if (rcx->rc.effective_uid != 0)
1126 1     {
1127 1         if (debugmode)
1128 1         {
1129 1             rbe_log_status(
1130 1                 0, "changing the uid to admin status");
1131 1         }
1132 1         rcx->rc.effective_uid = 0;
1133 1         rcx->rc.effective_username = "root";
1134 1         *changed = SET_ROOT;
1135 1     }
1136 1     else
1137 1     {
1138 1         if (debugmode)
1139 1         {
1140 1             rbe_log_status(
1141 1                 0, "effect uid = %d", rcx->rc.effective_uid);
1142 1         }
1143 1     }
1144 1 }
1145 1
1146 1 /*
1147 1 * The user does not have root access to the dest
1148 1 * client. But s/he is the system admin for the
1149 1 * source client. Therefore, we need to strip the
1150 1 * root stuff from the user during the recovery.
1151 1 */
1152 1 rcx->rc.recovery_flags ^= RC_RECPFLAG_DEST_SYSADMIN;
1153 1 if (rcx->rc.recovery_flags & RC_RECPFLAG_SOURCE_SYSADMIN)
1154 1 {
1155 1     if (debugmode)
1156 1     {
1157 1         rbe_log_status(
1158 1             0, "changing the uid to regular user status");
1159 1     }
1160 1 }

```

```

1160 3      rcx->rc_effective_uid = rcx->rc_human_uid;
1161 3      rcx->rc_effective_username = rcx->rc_human_username;
1162 3      *changed = SET_USER;
1163 2      }
1164 2      /* end of set_recovery_privileges() */
1165 1      }

```

```

1166      /*
1167      * Reset the user's identity. The user may have root access
1168      * on the destination client, but not on the source, and vice
1169      * versa.
1170      */
1171      static void
1172      reset_recovery_privileges(struct recover_context *rcx,
1173                               int changed)
1174      {
1175          if (changed == SET_ROOT)
1176          {
1177              if (debugmode)
1178              {
1179                  the_log_stack(
1180                      0, "resetting the uid to regular user status");
1181              }
1182              rcx->rc_effective_uid = rcx->rc_human_uid;
1183              rcx->rc_effective_username = rcx->rc_human_username;
1184          }
1185          else
1186          {
1187              if (debugmode)
1188              {
1189                  the_log_stack(0, "resetting the uid to sys admin status");
1190              }
1191              rcx->rc_effective_uid = 0;
1192              rcx->rc_effective_username = "root";
1193          }
1194          /* end of reset_recovery_privileges() */
1195      }

```

```

1202  /**
1203  **
1204  ** FUNCTION DESCRIPTION:
1205  **
1206  ** This function will start the workitem restore termination.
1207  **
1208  **
1209  ** INPUTS:
1210  **
1211  **   int auxproc_pid -- auxproc's pid.
1212  **
1213  ** RETURN VALUE:
1214  **
1215  **   none
1216  **
1217  ** SIDE EFFECTS:
1218  **
1219  **   auxproc sent the user1 signal, auxproc will send xcplogon the
1220  **   TERM signal to quit the restore, auxproc will wait until the
1221  **   restore terminates by waiting for the remote command's exit
1222  **   status.
1223  **
1224  **++
1225  **/
1226
1227 void
1228 QuitWorkItemRestore(int auxproc_pid)
1229 {

```

```

1230
1231  /**
1232  ** auxproc will now alert xcplogon by sending the
1233  ** TERM signal.
1234  ** This will give xcplogon the ability to
1235  ** clean up caplases and clean-up sockets.
1236  ** auxproc will commit suicide as a result of
1237  ** this signal.
1238  **/
1239
1240
1241

```

```

1242  /**
1243  ** Alert the auxproc that we want out.
1244  ** rexecpio etc should also die as a result
1245  ** of xc_read_or_die_xc() in rexc. The SIGNAL
1246  ** should not kill the auxproc itself, but only
1247  ** notify auxproc of the diminishing restore.
1248  **/
1249

```

```

1250  (void)kill(auxproc_pid, SIGUSR1);
1251
1252  if (debugmode)
1253  {
1254      the_log_state(0, "the %d quitting restore in process",
1255                  auxproc_pid);
1256  }
1257

```

```

1258  /**
1259  ** now that we have indirectly killed the xcplogon and alerted the
1260  ** auxproc,
1261  ** the signal from the auxproc will be picked up by the next
1262  ** routing, auxprocsig_handler, it will notify the user of the
1263  ** results and does the cleaning up.
1264  */
1265

```

```

1265  /**
1266  **
1267  ** returns;
1268  **
1269  **   QuitWorkItemRestore() */
1270

```

```

1250 /** GetAuxprocResults()
1251 **
1252 ** FUNCTION DESCRIPTION:
1253 **
1254 ** Inherited from auxprocapi_handler().
1255 **
1256 ** This routine is called when there is information appears
1257 ** in the aux-process. The aux-process starts xcplogon and
1258 ** is responsible for "listening" to status coming from xcplogon.
1259 ** When status comes back from xcplogon, the aux-process signals
1260 ** this process, which is trapped by the caller. And finally,
1261 ** this routine is called.
1262 **
1263 ** Args:
1264 ** (1) read int -- results file descriptor from auxproc.
1265 ** (0) results -- work item results.
1266 **
1267 ** RETURN VALUE:
1268 ** The number of local and remote status collected from fd.
1269 **
1270 ** SIDE EFFECTS:
1271 **
1272 ** none
1273 **
1274 **
1275 **/
1276
1277 int
1278 GetAuxprocResults(int read,
1279                   w_t *results, results *results)
1280 {
1281     int
1282     exit_stat;
1283     char
1284     *result;
1285     int
1286     c = 0;
1287     n_status_read = 0;
1288     int
1289     while ((n_status_read < 2) && (1 == fd_await_test(read)))
1290     {
1291         result = (char *) exit_stat;
1292         if (1 == pread_or_wait(read, &c, 1, auxproc_comm_warning))
1293             return 0;
1294     }
1295     if (debugmode)
1296     {
1297         the_log_state(0, "GetAuxprocResults() called: '%c', c);
1298     }
1299     if (c == 'R')
1300     {
1301         if (1 == fd_await_test(read))
1302         {
1303             /*
1304             * the 'R' command is the for the remote process status
1305             */
1306             n_read = auxread(read, 'R', sizeof(int), &exit_stat);
1307             if (c == n_status_read)
1308             {
1309                 result = (char *) exit_stat;
1310                 result -> remote_exit_status = exit_stat;
1311                 result -> remote_exit_set = TRUE;
1312                 n_status_read++;
1313             }
1314         }
1315     }
1316     return n_status_read;
1317 }

```

```

1312 }
1313 {
1314     if (debugmode)
1315     {
1316         the_log_state(
1317             0, "remote exit status obtained: %d", exit_stat);
1318     }
1319     else
1320     {
1321         the_log_state(
1322             0, "Internal error: remote exit status incomplete.");
1323         return -1;
1324     }
1325     }
1326     else if (c == 'r')
1327     {
1328         if (1 == fd_await_test(read))
1329         {
1330             /*
1331             * the 'r' command is the for the local process status
1332             */
1333             n_read = auxread(read, 'r', sizeof(int), &exit_stat);
1334             if (c == n_status_read)
1335             {
1336                 results -> local_exit_status = exit_stat;
1337                 results -> local_exit_set = TRUE;
1338                 n_status_read++;
1339                 if (debugmode)
1340                 {
1341                     the_log_state(
1342                         0, "local exit status obtained: %d", exit_stat);
1343                 }
1344             }
1345             else
1346             {
1347                 the_log_state(
1348                     0, "Internal error: local exit status incomplete.");
1349                 return -1;
1350             }
1351         }
1352         /* sleep (1) */
1353         while (1)
1354         {
1355             return n_status_read;
1356         }
1357     }
1358     return n_status_read;
1359 }
1360
1361 /* GetAuxprocResults() */
1362

```

```

1389 /*
1390  * KillWorkItemRestore()
1391  *
1392  * Kill the work item restore, keep in mind this
1393  * restore has done if the work item is not running
1394  * a restore.
1395  *
1396  * This routine also does the waitpid for auxproc.
1397  * The waitpid (ChildDone()) eliminates the duplicate auxproc
1398  * process.
1399  *
1400  * If the work item is running then one must do the
1401  * following.
1402  *
1403  * 1) Call KillWorkItemRestore()
1404  * 2) Wait for and read results from the cmd_from pipe.
1405  * 3) Call KillWorkItemRestore()
1406  *
1407  * Args:
1408  *   ap_pid -- auxproc pid.
1409  *   cmd_to -- auxproc cmd pipe.
1410  *
1411  * Return:
1412  *   int -- zero for success.
1413  */
1414 int
1415 KillWorkItemRestore(int ap_pid, int cmd_to)
1416 {
1417     int killRet;
1418     int apResult;
1419     char *apName=KIXPROCNAME;
1420     char *dataBuf = NULL;
1421     int ChildDoneRet;
1422
1423     killRet = kill(ap_pid, SIGTERM);
1424
1425     if (-1 == killRet)
1426     {
1427         rbe_log_stats(
1428             0, "Can't send signal to '%s': Pid: %d, error = %s\n",
1429             apName, ap_pid, strerror(errno));
1430         return -1;
1431     }
1432
1433     do
1434     {
1435         ChildDoneRet = ChildDone(ap_pid, &apResult);
1436         if (0 == ChildDoneRet) sleep (1);
1437     } while(0 == ChildDoneRet);
1438
1439     switch (ChildDoneRet)
1440     {
1441         /* -1 internal error, error is set.
1442          * 0 child still running.
1443          * 1 child exited.
1444          * 2 child signaled (no core)
1445          * 3 child signaled core file generated.
1446          * 4 child stopped.
1447          */
1448         case (-1):
1449             return -1;
1450         /* no break necessary */
1451     }
1452 }

```

```

1443 case(1):
1444     rbe_log_stats(0,
1445         "Sigterm did not bring down '%s': Pid: %d, "
1446         "It instead exited with = %d\n",
1447         apName, ap_pid, apResult);
1448     break;
1449
1450 case(2):
1451     case(3):
1452         if (SIGTERM != apResult)
1453         {
1454             rbe_log_stats(0,
1455                 "Sigterm did not bring down '%s': Pid: %d, "
1456                 "Process killed by signal = %s\n",
1457                 apName, ap_pid,
1458                 strsignal(apResult));
1459             break;
1460         }
1461
1462 case(4):
1463     rbe_log_stats(0,
1464         "Sigterm did not bring down '%s': Pid: %d, "
1465         "Instead stopped by signal = %s\n",
1466         apName, ap_pid,
1467         strsignal(apResult));
1468     break;
1469
1470 }
1471
1472 }
1473
1474 /* auxprocBack(cmd_to, 'q', 0, dataBuf, */
1475 return 0;
1476 }
1477 } /* KillWorkItemRestore() */

```



```

138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181

```

```

enum input_states
{next_state_ptr,
  boolenly,
  *skipping_leading_whitespace,
  int *parsepos,
  int *mappos};

static char *recover_size_prefix(struct auxproc_context *cxp);
static int *chr_direct_from(char **abost, unsigned short input,
  struct auxproc_context *cxp, char *lcount,
  char *remuser, char *cmd, int *fdp);

```

```

182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

```

/*
 * The auxiliary process(es) communicate with the main
 * process via a simple protocol run on a pair of pipes.
 * The parent writes commands to the auxiliary process.
 * The format of a command is:
 *
 * <cmd><data-len><data>
 *
 * where <cmd> is a one-byte command
 * <data-len> is an "int",
 * and indicates the number of <data> bytes
 * <data> is command-specific data.
 *
 * <data-len> may be zero, but must always be present. Therefore, the
 * minimum command "packet" is five bytes long.
 *
 * Result packets are written back to the parent. The packet format is
 * the same as the command format, though (obviously) the format
 * of the data in a result packet is usually different from
 * the format in a command packet.
 *
 * Communications are assumed to be error-free. In other
 * words, pipes are assumed to work correctly.
 */
static int attr_ec; /* count of SIGUSR1 (ARMY signals) */

```



```

183
184 {
185     int main(int argc, char *argv[])
186     {
187         char *auxproc_initial_delay_str = NULL;
188         int auxproc_initial_delay_int = 0;
189         if (argc != 9)
190             exit(1);
191         auxproc_initial_delay_str = getenv("MUXPROC_INITIAL_DELAY");
192         if (NULL != auxproc_initial_delay_str)
193             auxproc_initial_delay_int = atoi(auxproc_initial_delay_str);
194         sleep(auxproc_initial_delay_int);
195     }
196
197     debugmode = atoi(argv[6]);
198
199     do_auxproc(atoi(argv[1]), /* auxproc ordinate */
200               atoi(argv[2]), /* cmd to auxproc pipe */
201               atoi(argv[3]), /* cmd from auxproc pipe */
202               atoi(argv[4]), /* proc from auxproc pipe */
203               argv[0], /* program name */
204               argv[5], /* socket host name */
205               atoi(argv[8])); /* socket port */
206
207     return(0); /* Can we get here ?? */
208 }
209
210
211 } /* End main() */

```

```

215 /*
216  * Worker-Dee loop.
217  * Read instructions on r_fd.
218  * Write results to w_fd.
219  */
220
221 int membs_rcvd = 0;
222
223 void
224 do_auxproc(int procnum,
225            int r_fd,
226            int w_fd,
227            int r_bulk_fd,
228            int w_bulk_fd,
229            int w_prog_fd,
230            char *sockhost,
231            int sockport)
232 {
233     struct auxproc_context
234     {
235         struct servent
236             *serv;
237         struct servent
238             *rotation_size;
239         int
240             index_fd;
241     };
242     struct auxproc_context
243     {
244         struct servent
245             *serv;
246         struct servent
247             *rotation_size;
248         int
249             index_fd;
250     };
251     /* close all file descriptors that we do not need
252     * and add the ones passed to us from the restore
253     * engine. So we can close anything above stderr and anything
254     * that is not equal to what was passed in
255     */
256     for(index_fd = 3; MAX_FD > index_fd; index_fd++)
257     {
258         if ((index_fd == r_fd) ||
259             (index_fd == w_fd) ||
260             (index_fd == r_bulk_fd) ||
261             (index_fd == w_bulk_fd) ||
262             (index_fd == w_prog_fd))
263             continue;
264         (void)close(index_fd);
265     }
266
267     memset(&ctx, 0, sizeof(struct auxproc_context));
268     /* Prepare sigaction parameters
269     */
270     sigemptyset(&empty_set);
271
272     /* ignore mid keyboard interrupts (^C); parent handles
273     */
274     sigignore(SIGINT, SIG_IGN, &empty_set, &SA_RESTART);
275
276     /* SIGINT used to get our attention when the parent
277     * process really wants us to stop what we are doing.
278     */
279     ab_set_signal_handler(

```

```

330 1          struct signal_handler, empty_set, E_SH_RESTART);
331 1          eb_set_signal_handler(
332 1              SIGTERM, signal_handler, empty_set, E_SH_RESTART);
333 1
334 1          /*
335 1             * Arguments passed are collected together in
336 1             * a struct, and are only because that makes
337 1             * it easier to add/change arguments in the future
338 1             * (didn't the structure rather than diddle a zillion
339 1             * calls in the switch statement).
340 1             */
341 1
342 1          ctx.ap.argv_auxnum = procnum;
343 1          ctx.ap.rc_fd = rc_fd;
344 1          ctx.ap.w_prog_fd = w_prog_fd;
345 1          ctx.ap.w_prog_fd = w_prog_fd;
346 1          ctx.ap.sockethost = sockethost;
347 1          ctx.ap.sockeport = sockeport;
348 1
349 1          /*
350 1             * We'll need the shell/tcp service later, if
351 1             * can't find it now we might as well complain now.
352 1             */
353 1          if (ctx.ap.have_shelltcp_port == 0)
354 1              if ((sp = getentrybyname("shell", "tcp")) != NULL)
355 1                  ctx.ap.have_shelltcp_port = 0;
356 1          if (ctx.ap.have_shelltcp_port != (ushort)(LSP->port);
357 1              ctx.ap.have_shelltcp_port = 1;
358 1          )
359 1              else
360 1                  /*
361 1                     * Write the stringing
362 1                     */
363 1                  WriteStringing(
364 1                      ctx.ap.w_prog_fd, ERRERRNOCSG_AUXPROC_ERROR, 0,
365 1                      "Error: Cannot find shell/tcp service, but
366 1                      * browsing of catalogs may continue, but
367 1                      * the 'verify' command will not work.");
368 1          }
369 1          if (NULL == ctx.ap.config)
370 1              if (NULL == (ctx.ap.config = malloc(sizeof(
371 1                  struct tbc_configs))))
372 1                  /*
373 1                     * We would prefer to log this stuff, but we
374 1                     * have not opened logging yet.
375 1                     */
376 1                  WriteStringing(
377 1                      ctx.ap.w_prog_fd, ERRERRNOCSG_AUXPROC_ERROR, 0,
378 1                      "Could not allocate memory in
379 1                      do_auxproc");
380 1          }
381 1          exit(1);
382 1      }
383 1      if (tbc_parse_config(
384 1          NULL, /* use the default name */ , &ctx.ap.config,
385 1          RBC_PHASE_DO_NOT_RESERVE |
386 1          RBC_PHASE_APPLY) != 0)
387 1          /*
388 1             * We would prefer to log this stuff, but we
389 1             * have not opened logging yet.
390 1             */
391 1          ;
392 1
393 1
394 1
395 1
396 1
397 1
398 1
399 1
400 1

```

```

337 3          rec_ap_log_cmd(SUB_CMD_PHASE_CFG, NULL);
338 3          WriteStringing(
339 3              ctx.ap.w_prog_fd, ERRERRNOCSG_AUXPROC_ERROR, 0,
340 3              "Auxproc -- Cannot parse configuration
341 3              file");
342 3          }
343 3          exit(1);
344 3          if ((errnum = eb_path_init()) != 0)
345 3              if (exit(1);
346 3                  )
347 3                  (void)the_log_init("auxproc");
348 3          }
349 3          /*
350 3             * Determine the "recovery_log" file rotation size.
351 3             * Use a default unless this was specified in the config file.
352 3             */
353 3          rotation_size = RANCOVER_LOGSIZE;
354 3          if ((ctx.ap.config != (struct tbc_config *)NULL)
355 3              && (ctx.ap.config->logfile_filename != NULL))
356 3              rotation_size = ctx.ap.config->logfile_rotation_size;
357 3          (void)the_log_add(status_logging, eb_recovery_logpath, LOGT_FILE,
358 3              rotation_size, &logging_channel);
359 3          /*
360 3             * The actual worker does loop
361 3             */
362 3          for (;;)
363 3              {
364 3                  char c;
365 3                  int decalen;
366 3                  #define SMALL_DATALEN 128
367 3                  char *data;
368 3                  /*
369 3                     * read command byte
370 3                     */
371 3                  preend_or_disconnect(&c, 1, &exit);
372 3                  /*
373 3                     * read decalen
374 3                     */
375 3                  preend_or_disconnect(&decalen,
376 3                      sizeof(decalen),
377 3                      &exit);
378 3                  /*
379 3                     * We do the read here for simple commands that do not
380 3                     * take much data.
381 3                     */
382 3                  if (decalen > SMALL_DATALEN || decalen == 0)
383 3                      {
384 3                          data = NULL;
385 3                      }
386 3                  else
387 3                      {
388 3                          data = malloc(decalen);
389 3                          if (data == NULL)
390 3                              {
391 3                                  /*
392 3                                     * We do the read here for simple commands that do not
393 3                                     * take much data.
394 3                                     */
395 3                                  if (decalen > SMALL_DATALEN || decalen == 0)
396 3                                      {
397 3                                          data = NULL;
398 3                                      }
399 3                                  else
400 3                                      {

```

```

401.3 {
402.3     press_or_die(r_fd, buf, datalen, _exit);
403.3     data = buf;
404.2 }
405.2
406.2 /* rcmd: rcmd:
407.2     ctx.ap.datalen = datalen;
408.2     ctx.ap.data = data;
409.2     ctx.ap.resultlen = 0;
410.2     ctx.ap.resultdata = NULL;
411.2 */
412.2 /* NOTE: This switch statement is in alpha-order
413.2     (case-folded) to make it easier for humans
414.2     to find entries (at least, it was in alpha-order
415.2     when I wrote this comment).
416.2 */
417.2
418.2 switch (c)
419.2 {
420.2     /*
421.2     /* 'P'
422.2     /* Ring command. Used for debugging.
423.2     */
424.2     /*
425.2     case 'P':
426.2         ctx.ap.resultlen = ctx.ap.datalen;
427.2         ctx.ap.resultdata = ctx.ap.data;
428.2         break;
429.2     /*
430.2     /* 'Q'
431.2     /* Quit command.
432.2     */
433.2
434.2 case 'Q':
435.2     rbe_close_log(logging_channel);
436.2     _exit(0);
437.2
438.2 /*
439.2 /* 'R'
440.2 /* rcmd filter: fire up a process, typically xloggen,
441.2 /* with output from the process going to an rcmd
442.2 /* connection.
443.2 */
444.2 case 'R':
445.2     z_rcmdfilter(&ctx);
446.2     break;
447.2
448.2 /*
449.2 /* 'X'
450.2 /* Exec separate process version of do_auxproc.
451.2 /* Used for debugging.
452.2 /* No value returned to parent.
453.2 /* If exec fails, auxproc dies.
454.2 */
455.2 case 'X':
456.2     z_exec_separate_auxproc(&ctx, Xname);
457.2     _exit(2);
458.2     break;
459.2
460.2 default:
461.2     rbe_log_stats(
462.2         0, "auxproc -- invalid command in do_auxproc");
463.2
464.3 }
465.3

```

```

466.3     _exit(1);
467.3     break;
468.2 }
469.2 /* end of switch */
470.2
471.2 /*
472.2 /* Simple commands set ap_resultlen and ap_resultdata
473.2 /* and we push the results to the parent here.
474.2 */
475.2 if (ctx.ap.resultlen >= 0)
476.2 {
477.2     /*
478.2     /* write_or_die(r_fd, kc, 1, _exit);
479.2     /* write_or_die(r_fd, (char *)(&ctx.ap.resultlen,
480.2     /* sizeof ctx.ap.resultlen, _exit);
481.2     /* write_or_die(r_fd, ctx.ap.resultdata, _exit);
482.2     /*
483.2     /*
484.2     /*
485.2     /*
486.2     /*
487.2     /*
488.2     /*
489.2     /*
490.2     /*
491.2     /*
492.2     /*

```

```

434 */
435 * Fire up a process with stdin from parent and output to rcmd.
436 * Protocol traffic between "parent" (main process) and us:
437 *
438 * 'P' parent --> auxproc contains local & remote cmd info
439 * 'R' auxproc --> parent returns 'setup' result, see below
440 * 'R' auxproc --> parent returns remote exit status (i int)
441 * 'r' auxproc --> parent returns local exit status (i int)
442 *
443 * The '0' result packet describes the results of setting things up.
444 * It contains the following information:
445 *
446 * success/failure: int [ 0 is success,
447 *                       non-zero is a failure code ]
448 * errnum : int [ 0 if no applicable error code ]
449 * pid : int [ process ID of local xplrogen proc ]
450 *
451 * maxlen : int [ length, in bytes,
452 *                 of following str ]
453 *
454 * errmsg : string [ string describing failure ]
455 *
456 * If things were set up successfully, then two integer zero
457 * values, a non-zero pid, and one more zero (maxlen) are
458 * sent in the '0' packet.
459 *
460 * The defined failure codes are:
461 *
462 * 790
463 * The 'R' result packet will not be sent if the '0' result
464 * indicates that an error occurred. The '0' and 'r' result
465 * packets are always sent.
466 *
467 * Parent passes the following in the 'r' packet <data>:
468 *
469 * (string) rcmd_hostname
470 *
471 * (string) rcmd_hostname
472 *
473 * (string) rcmd_username
474 *
475 * (string) rcmd_password
476 *
477 * (int) rcmd_val
478 *
479 * (int) filter-cmd-id-info (explained below)
480 *
481 * (int) future-flags (flags for future packs; always zero now)
482 *
483 * (string) filter-cmd
484 *
485 * (int) filter-cmd-arg0
486 *
487 * (string) filter-cmd-arg0
488 *
489 * (string) ...
490 *
491 * (string) filter-cmd-argN
492 *
493 * (int) db API socket host name
494 *
495 * (string) filter-cmd-argN
496 *
497 * (string) filter-cmd-argN
498 *
499 * If filter-cmd-id-info is -1, then stdout on the filter cmd
500 * is set up to go to the rcmd.
501 *
502 * Otherwise, a separate file descriptor (neither stdout nor stderr)
503 * is set up to go to the rcmd, and the argv element indicated
504 * by filter-cmd-argN is replaced with the file descriptor for passing
505 * the file descriptor number to the filter process.
506 *
507 * For example, if filter-cmd-id-info is 1, then filter-cmd-argN1
508 * should be a sprintf format string, which will be given to
509 * sprintf along with one integer to pass the file descriptor number
510 * to the filter command.
511 *
512 * Items which are (string) are '0' terminated.
513

```

```

555 * no more than 100 argv strings for filter-cmd
556 * no more than 1000 local bytes of auxproc data
557 * no more than 100 bytes of filter-cmd-id-info argv
558 */
559
560 struct rcmd_pkto_info
561 {
562     int failcode;
563     int errnum;
564     int pid;
565     int maxlen;
566     /* variable length char string message follows */
567 };
568
569 /*
570 * Function to build command line prefix for adding environment
571 * for log truncation
572 */
573
574 static char *
575 recover_size_prefix(struct auxproc_context *cxp)
576 {
577     static char buff[128];
578
579     if (cxp->ap_config->ctrlfile.rotation_size != NO_ROTATION)
580     {
581         sprintf(buff, "DB_MAX_CLIENT_LOG_SIZE=%i",
582             cxp->ap_config->ctrlfile.rotation_size);
583     }
584     else
585     {
586         memset(buff, 0, sizeof(buff));
587     }
588
589     return buff;
590 }
591
592 /* recover_size_prefix */

```

```

597 struct avoid
598 {
599     char
600     *cmd;
601     char
602     *human_name;
603     struct rcmd_pkt0 pkt0;
604     char
605     *pkt0_error = "";
606     static int
607     f_resultIndex;
608     int
609     f_resultIndex;
610     int
611     filter_and_foInfo;
612     char
613     *filter_cmd;
614     char
615     *method;
616     int
617     descripto=0;
618     int
619     socket_port;
620     char
621     *socket_file;
622     FILE
623     *logfile[200];
624     FILE
625     *infofile;
626     /* items for socket info file --- */
627     static char
628     readnode[] = "r";
629     char
630     *stat[200];
631     FILE
632     *filename[EX_MAXPATHLEN];
633     char
634     *workitem;
635     int
636     rcmd_scheter = -1;
637     int
638     *rcmd_fdi;
639     int
640     rcmd_fds[2];
641     int
642     fd;
643     int
644     fd2;
645     void
646     *result;
647     char
648     *fdagov[100];
649     char
650     *fdagov_and_argv[100];
651     char
652     *descrip[10000];
653     char
654     *fdagov[200];
655     /* for command with sel parameter */
656     char
657     holdbuff[1024];
658     /* hold fdclient .. command */
659     RRC_WORKITEM *pwi;
660     RRC_WORKITEM *pwi2;
661     RRC_WORKITEM *pwi3;
662     RRC_WORKITEM *pwi4;
663     RRC_WORKITEM *pwi5;
664     RRC_WORKITEM *pwi6;
665     RRC_WORKITEM *pwi7;
666     RRC_WORKITEM *pwi8;
667     RRC_WORKITEM *pwi9;
668     RRC_WORKITEM *pwi10;
669     RRC_WORKITEM *pwi11;
670     RRC_WORKITEM *pwi12;
671     RRC_WORKITEM *pwi13;
672     RRC_WORKITEM *pwi14;
673     RRC_WORKITEM *pwi15;
674     RRC_WORKITEM *pwi16;
675     RRC_WORKITEM *pwi17;
676     RRC_WORKITEM *pwi18;
677     RRC_WORKITEM *pwi19;
678     RRC_WORKITEM *pwi20;
679     RRC_WORKITEM *pwi21;
680     RRC_WORKITEM *pwi22;
681     RRC_WORKITEM *pwi23;
682     RRC_WORKITEM *pwi24;
683     RRC_WORKITEM *pwi25;
684     RRC_WORKITEM *pwi26;
685     RRC_WORKITEM *pwi27;
686     RRC_WORKITEM *pwi28;
687     RRC_WORKITEM *pwi29;
688     RRC_WORKITEM *pwi30;
689     RRC_WORKITEM *pwi31;
690     RRC_WORKITEM *pwi32;
691     RRC_WORKITEM *pwi33;
692     RRC_WORKITEM *pwi34;
693     RRC_WORKITEM *pwi35;
694     RRC_WORKITEM *pwi36;
695     RRC_WORKITEM *pwi37;
696     RRC_WORKITEM *pwi38;
697     RRC_WORKITEM *pwi39;
698     RRC_WORKITEM *pwi40;
699     RRC_WORKITEM *pwi41;
700     RRC_WORKITEM *pwi42;
701     RRC_WORKITEM *pwi43;
702     RRC_WORKITEM *pwi44;
703     RRC_WORKITEM *pwi45;
704     RRC_WORKITEM *pwi46;
705     RRC_WORKITEM *pwi47;
706     RRC_WORKITEM *pwi48;
707     RRC_WORKITEM *pwi49;
708     RRC_WORKITEM *pwi50;
709     RRC_WORKITEM *pwi51;
710     RRC_WORKITEM *pwi52;
711     RRC_WORKITEM *pwi53;
712     RRC_WORKITEM *pwi54;
713     RRC_WORKITEM *pwi55;
714     RRC_WORKITEM *pwi56;
715     RRC_WORKITEM *pwi57;
716     RRC_WORKITEM *pwi58;
717     RRC_WORKITEM *pwi59;
718     RRC_WORKITEM *pwi60;
719     RRC_WORKITEM *pwi61;
720     RRC_WORKITEM *pwi62;
721     RRC_WORKITEM *pwi63;
722     RRC_WORKITEM *pwi64;
723     RRC_WORKITEM *pwi65;
724     RRC_WORKITEM *pwi66;
725     RRC_WORKITEM *pwi67;
726     RRC_WORKITEM *pwi68;
727     RRC_WORKITEM *pwi69;
728     RRC_WORKITEM *pwi70;
729     RRC_WORKITEM *pwi71;
730     RRC_WORKITEM *pwi72;
731     RRC_WORKITEM *pwi73;
732     RRC_WORKITEM *pwi74;
733     RRC_WORKITEM *pwi75;
734     RRC_WORKITEM *pwi76;
735     RRC_WORKITEM *pwi77;
736     RRC_WORKITEM *pwi78;
737     RRC_WORKITEM *pwi79;
738     RRC_WORKITEM *pwi80;
739     RRC_WORKITEM *pwi81;
740     RRC_WORKITEM *pwi82;
741     RRC_WORKITEM *pwi83;
742     RRC_WORKITEM *pwi84;
743     RRC_WORKITEM *pwi85;
744     RRC_WORKITEM *pwi86;
745     RRC_WORKITEM *pwi87;
746     RRC_WORKITEM *pwi88;
747     RRC_WORKITEM *pwi89;
748     RRC_WORKITEM *pwi90;
749     RRC_WORKITEM *pwi91;
750     RRC_WORKITEM *pwi92;
751     RRC_WORKITEM *pwi93;
752     RRC_WORKITEM *pwi94;
753     RRC_WORKITEM *pwi95;
754     RRC_WORKITEM *pwi96;
755     RRC_WORKITEM *pwi97;
756     RRC_WORKITEM *pwi98;
757     RRC_WORKITEM *pwi99;
758     RRC_WORKITEM *pwi100;
759     RRC_WORKITEM *pwi101;
760     RRC_WORKITEM *pwi102;
761     RRC_WORKITEM *pwi103;
762     RRC_WORKITEM *pwi104;
763     RRC_WORKITEM *pwi105;
764     RRC_WORKITEM *pwi106;
765     RRC_WORKITEM *pwi107;
766     RRC_WORKITEM *pwi108;
767     RRC_WORKITEM *pwi109;
768     RRC_WORKITEM *pwi110;
769     RRC_WORKITEM *pwi111;
770     RRC_WORKITEM *pwi112;
771     RRC_WORKITEM *pwi113;
772     RRC_WORKITEM *pwi114;
773     RRC_WORKITEM *pwi115;
774     RRC_WORKITEM *pwi116;
775     RRC_WORKITEM *pwi117;
776     RRC_WORKITEM *pwi118;
777     RRC_WORKITEM *pwi119;
778     RRC_WORKITEM *pwi120;
779     RRC_WORKITEM *pwi121;
780     RRC_WORKITEM *pwi122;
781     RRC_WORKITEM *pwi123;
782     RRC_WORKITEM *pwi124;
783     RRC_WORKITEM *pwi125;
784     RRC_WORKITEM *pwi126;
785     RRC_WORKITEM *pwi127;
786     RRC_WORKITEM *pwi128;
787     RRC_WORKITEM *pwi129;
788     RRC_WORKITEM *pwi130;
789     RRC_WORKITEM *pwi131;
790     RRC_WORKITEM *pwi132;
791     RRC_WORKITEM *pwi133;
792     RRC_WORKITEM *pwi134;
793     RRC_WORKITEM *pwi135;
794     RRC_WORKITEM *pwi136;
795     RRC_WORKITEM *pwi137;
796     RRC_WORKITEM *pwi138;
797     RRC_WORKITEM *pwi139;
798     RRC_WORKITEM *pwi140;
799     RRC_WORKITEM *pwi141;
800     RRC_WORKITEM *pwi142;
801     RRC_WORKITEM *pwi143;
802     RRC_WORKITEM *pwi144;
803     RRC_WORKITEM *pwi145;
804     RRC_WORKITEM *pwi146;
805     RRC_WORKITEM *pwi147;
806     RRC_WORKITEM *pwi148;
807     RRC_WORKITEM *pwi149;
808     RRC_WORKITEM *pwi150;
809     RRC_WORKITEM *pwi151;
810     RRC_WORKITEM *pwi152;
811     RRC_WORKITEM *pwi153;
812     RRC_WORKITEM *pwi154;
813     RRC_WORKITEM *pwi155;
814     RRC_WORKITEM *pwi156;
815     RRC_WORKITEM *pwi157;
816     RRC_WORKITEM *pwi158;
817     RRC_WORKITEM *pwi159;
818     RRC_WORKITEM *pwi160;
819     RRC_WORKITEM *pwi161;
820     RRC_WORKITEM *pwi162;
821     RRC_WORKITEM *pwi163;
822     RRC_WORKITEM *pwi164;
823     RRC_WORKITEM *pwi165;
824     RRC_WORKITEM *pwi166;
825     RRC_WORKITEM *pwi167;
826     RRC_WORKITEM *pwi168;
827     RRC_WORKITEM *pwi169;
828     RRC_WORKITEM *pwi170;
829     RRC_WORKITEM *pwi171;
830     RRC_WORKITEM *pwi172;
831     RRC_WORKITEM *pwi173;
832     RRC_WORKITEM *pwi174;
833     RRC_WORKITEM *pwi175;
834     RRC_WORKITEM *pwi176;
835     RRC_WORKITEM *pwi177;
836     RRC_WORKITEM *pwi178;
837     RRC_WORKITEM *pwi179;
838     RRC_WORKITEM *pwi180;
839     RRC_WORKITEM *pwi181;
840     RRC_WORKITEM *pwi182;
841     RRC_WORKITEM *pwi183;
842     RRC_WORKITEM *pwi184;
843     RRC_WORKITEM *pwi185;
844     RRC_WORKITEM *pwi186;
845     RRC_WORKITEM *pwi187;
846     RRC_WORKITEM *pwi188;
847     RRC_WORKITEM *pwi189;
848     RRC_WORKITEM *pwi190;
849     RRC_WORKITEM *pwi191;
850     RRC_WORKITEM *pwi192;
851     RRC_WORKITEM *pwi193;
852     RRC_WORKITEM *pwi194;
853     RRC_WORKITEM *pwi195;
854     RRC_WORKITEM *pwi196;
855     RRC_WORKITEM *pwi197;
856     RRC_WORKITEM *pwi198;
857     RRC_WORKITEM *pwi199;
858     RRC_WORKITEM *pwi200;
859     RRC_WORKITEM *pwi201;
860     RRC_WORKITEM *pwi202;
861     RRC_WORKITEM *pwi203;
862     RRC_WORKITEM *pwi204;
863     RRC_WORKITEM *pwi205;
864     RRC_WORKITEM *pwi206;
865     RRC_WORKITEM *pwi207;
866     RRC_WORKITEM *pwi208;
867     RRC_WORKITEM *pwi209;
868     RRC_WORKITEM *pwi210;
869     RRC_WORKITEM *pwi211;
870     RRC_WORKITEM *pwi212;
871     RRC_WORKITEM *pwi213;
872     RRC_WORKITEM *pwi214;
873     RRC_WORKITEM *pwi215;
874     RRC_WORKITEM *pwi216;
875     RRC_WORKITEM *pwi217;
876     RRC_WORKITEM *pwi218;
877     RRC_WORKITEM *pwi219;
878     RRC_WORKITEM *pwi220;
879     RRC_WORKITEM *pwi221;
880     RRC_WORKITEM *pwi222;
881     RRC_WORKITEM *pwi223;
882     RRC_WORKITEM *pwi224;
883     RRC_WORKITEM *pwi225;
884     RRC_WORKITEM *pwi226;
885     RRC_WORKITEM *pwi227;
886     RRC_WORKITEM *pwi228;
887     RRC_WORKITEM *pwi229;
888     RRC
```

656 2	if (data == NULL)	681 1	human_name = data;	701 1	filter_cmd = data;	727 1	for (i = 0; i < filter_cmd_argc; i++)	743 1	filter_cmd_argv[filter_cmd_argc] = NULL;	769 1	/*	795 1	/*	821 1	human_name = data;	847 1	/*	873 1	/*	899 1	/*	925 1	/*	951 1	/*	977 1	/*	1003 1	/*	1029 1	/*	1055 1	/*	1081 1	/*	1107 1	/*	1133 1	/*	1159 1	/*	1185 1	/*	1211 1	/*	1237 1	/*	1263 1	/*	1289 1	/*	1315 1	/*	1341 1	/*	1367 1	/*	1393 1	/*	1419 1	/*	1445 1	/*	1471 1	/*	1497 1	/*	1523 1	/*	1549 1	/*	1575 1	/*	1601 1	/*	1627 1	/*	1653 1	/*	1679 1	/*	1705 1	/*	1731 1	/*	1757 1	/*	1783 1	/*	1809 1	/*	1835 1	/*	1861 1	/*	1887 1	/*	1913 1	/*	1939 1	/*	1965 1	/*	1991 1	/*	2017 1	/*	2043 1	/*	2069 1	/*	2095 1	/*	2121 1	/*	2147 1	/*	2173 1	/*	2199 1	/*	2225 1	/*	2251 1	/*	2277 1	/*	2303 1	/*	2329 1	/*	2355 1	/*	2381 1	/*	2407 1	/*	2433 1	/*	2459 1	/*	2485 1	/*	2511 1	/*	2537 1	/*	2563 1	/*	2589 1	/*	2615 1	/*	2641 1	/*	2667 1	/*	2693 1	/*	2719 1	/*	2745 1	/*	2771 1	/*	2797 1	/*	2823 1	/*	2849 1	/*	2875 1	/*	2901 1	/*	2927 1	/*	2953 1	/*	2979 1	/*	3005 1	/*	3031 1	/*	3057 1	/*	3083 1	/*	3109 1	/*	3135 1	/*	3161 1	/*	3187 1	/*	3213 1	/*	3239 1	/*	3265 1	/*	3291 1	/*	3317 1	/*	3343 1	/*	3369 1	/*	3395 1	/*	3421 1	/*	3447 1	/*	3473 1	/*	3499 1	/*	3525 1	/*	3551 1	/*	3577 1	/*	3603 1	/*	3629 1	/*	3655 1	/*	3681 1	/*	3707 1	/*	3733 1	/*	3759 1	/*	3785 1	/*	3811 1	/*	3837 1	/*	3863 1	/*	3889 1	/*	3915 1	/*	3941 1	/*	3967 1	/*	3993 1	/*	4019 1	/*	4045 1	/*	4071 1	/*	4097 1	/*	4123 1	/*	4149 1	/*	4175 1	/*	4201 1	/*	4227 1	/*	4253 1	/*	4279 1	/*	4305 1	/*	4331 1	/*	4357 1	/*	4383 1	/*	4409 1	/*	4435 1	/*	4461 1	/*	4487 1	/*	4513 1	/*	4539 1	/*	4565 1	/*	4591 1	/*	4617 1	/*	4643 1	/*	4669 1	/*	4695 1	/*	4721 1	/*	4747 1	/*	4773 1	/*	4799 1	/*	4825 1	/*	4851 1	/*	4877 1	/*	4903 1	/*	4929 1	/*	4955 1	/*	4981 1	/*	5007 1	/*	5033 1	/*	5059 1	/*	5085 1	/*	5111 1	/*	5137 1	/*	5163 1	/*	5189 1	/*	5215 1	/*	5241 1	/*	5267 1	/*	5293 1	/*	5319 1	/*	5345 1	/*	5371 1	/*	5397 1	/*	5423 1	/*	5449 1	/*	5475 1	/*	5501 1	/*	5527 1	/*	5553 1	/*	5579 1	/*	5605 1	/*	5631 1	/*	5657 1	/*	5683 1	/*	5709 1	/*	5735 1	/*	5761 1	/*	5787 1	/*	5813 1	/*	5839 1	/*	5865 1	/*	5891 1	/*	5917 1	/*	5943 1	/*	5969 1	/*	5995 1	/*	6021 1	/*	6047 1	/*	6073 1	/*	6099 1	/*	6125 1	/*	6151 1	/*	6177 1	/*	6203 1	/*	6229 1	/*	6255 1	/*	6281 1	/*	6307 1	/*	6333 1	/*	6359 1	/*	6385 1	/*	6411 1	/*	6437 1	/*	6463 1	/*	6489 1	/*	6515 1	/*	6541 1	/*	6567 1	/*	6593 1	/*	6619 1	/*	6645 1	/*	6671 1	/*	6697 1	/*	6723 1	/*	6749 1	/*	6775 1	/*	6801 1	/*	6827 1	/*</
-------	-------------------	-------	--------------------	-------	--------------------	-------	---------------------------------------	-------	--	-------	----	-------	----	-------	--------------------	-------	----	-------	----	-------	----	-------	----	-------	----	-------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	----	--------	------

Mon Oct 13 16:00:37 2008	z_cmffilter	Page 81 of 134
720 1	data -= sizeof (int);	
722 1	/*	
723 1	extract db API socket host name	
724 1	*/	
725 1	socket_file = data;	
727 1	data -= strlen(socket_file) + 1;	
729 1	/*	
730 1	extract filspec	
731 1	*/	
733 1	filspec = data;	
734 1	data -= strlen(filspec) + 1;	
736 1	/*	
737 1	extract workitem	
738 1	*/	
740 1	workitem = data;	
741 1	exp->ap_workitem = workitem;	
742 1	data -= strlen(workitem) + 1;	
744 1	pkto->aiinode = 0;	
745 1	pkto->extrem = 0;	
746 1	pkto->pid = -1;	
747 1	pkto->maglen = -1;	
748 1	pkto->extext = "";	
750 1	/*	
751 1	locate work item in config info	
752 1	*/	
754 1	if (NULL == exp->ap_config)	
755 2	{	
756 2	if (NULL == (exp->ap_config = malloc(sizeof(struct the_config)))
757 3	{	
758 3	the_log_status(
759 3	exit(1);	
760 2	0, "Could not allocate memory in z_cmffilter!");	
761 2	if (rbc_parse_config(
762 2	NULL, /*use the default name */ , exp->ap_config,	RBC_PARSE_DO_NOT_PRESERVE
	RBC_PARSE_APPLY) != 0)	
764 3	{	
765 3	rbc_apl_log_cm(CM_CMD_NO_PARSE_CFG, NULL);	
766 3	the_log_status(
767 3	exit(1);	
768 2	0, "auxproc -- Cannot parse configuration file!");	
769 1	}	
771 1	for (pwg = exp->ap_config->pgrouplist; ; pwg = pwg->next)	
772 2	{	
773 2	if (pwg == (RBC_WORKGROUP *)NULL)	
774 3	{	
775 3	char cmn_err_msg(256);	
777 3	the_log_status(
778 3	0, "\n ** No work item name \"%s\" in configuration file",	exp->ap_workitem);
779 3	apline(cm_err_msg,	

Mon Oct 13 16:00:37 2008	z_cmffilter	Page 82 of 134
780 3		
781 3	/*	
782 3	exp->ap_workitem;	
783 3	rbc_apl_log_cm(CM_CMD_NO_PARSE_CFG, cmn_err_msg);	
784 3	the_log_status(0, "Cannot continue without a work item!");	
785 3	exit(1);	
786 2	}	
787 2	for (pwi = pwg->pwlist; NULL != pwi; pwi = pwi->next)	
788 3	{	
789 3	if (0 == strcmp(pwi->name, exp->ap_workitem))	
790 4	{	
791 4	goto gotit;	
792 3	}	
793 2		
794 1		
796 1	gotit:	
797 1	/*	
798 1	connect by direct or rsh methods	
799 1	*/	
801 1	if ((method = rbc_getmethod(&cmn_errf(0), NULL, NULL)) == NULL)	
802 2	{	
803 2	writesStringing(
804 2	exp->ap_w_prog_fd, EDERRPROGMSG_AUXPROC_WARNING, 0,	
	"Unable to get connection method to host \"%s\"."	
	/*defaulting to \"rsh\" method*/	
	cmn_errf(0);	
806 2	method = "rsh";	
807 2		
808 1	}	
810 1	/*	
811 1	now check for a work item override of the connection method	
812 1	*/	
814 1	if (CNCN_RSH != pwi->xconnection_type) /* if other than rsh */	
815 2	{	
816 2	switch (pwi->xconnection_type)	
817 3	{	
818 3	case CNCN_RSH:	
819 3	method = "rsh";	
820 3	break;	
822 3	case CNCN_EMLINK:	
823 3	method = "emlink";	
824 3	break;	
826 3	case CNCN_DIRECT:	
827 3	method = "direct";	
828 3	break;	
830 3	case CNCN_SOCKET:	
831 3	method = "socket";	
832 3	break;	
834 3	case CNCN_NETWORK:	
835 3	method = "network";	
836 3	break;	
838 3	default:	
839 3	method = "???";	
840 3	break;	
841 2	}	

```

863 2         if (debugmode)
864 3             {
865 4                 rpe_log_statel(
866 5                     0, "using connection method '%s', due to work item '%s' override",
867 6                     method, pwi->name);
868 7             }
869 8         if (NULL != pwi && pwi->recovery_init) /* if a recovery init
870 9             command specified */
871 10             {
872 11                 (void)system(pwi->recovery_init);
873 12             }
874 13         /* OSGroup34932 -- Workaround for STPLose() behavioral defect
875 14             If pwi->ssl_groupname is not NULL, we are going to be using
876 15             SP for the restore.
877 16             Shut off the SSL (STPL) exit handler.
878 17             Environment variable is used here to the call to
879 18             CPU_mustexit can be avoided, if desired.
880 19             */
881 20         if (NULL == getenv("RMXPROC_DONT_PERFORM_CULMTEXTIT"))
882 21             {
883 22                 if (NULL != pwi->ssl_groupname)
884 23                     {
885 24                         if (NULL != getenv("RMX_SPL_LOGGING"))
886 25                             {
887 26                                 if (NULL != getenv("RMX_SPL_LOGGING"))
888 27                                     {
889 28                                         (void) putenv("STPL_EXITLOG_SHARE=OFF");
890 29                                     }
891 30                                 }
892 31                             }
893 32                     }
894 33         if (0 == strcmp(method, "direct"))
895 34             /*
896 35              * direct connection method here
897 36              */
898 37             {
899 38                 if (debugmode)
900 39                     {
901 40                         The_log_statel(0, "Invoking cmd with %s, %s, %s\n",
902 41                             cmd_stuff[0], cmd_stuff[1],
903 42                             cmd_stuff[2], cmd_stuff[3]);
904 43                     }
905 44                 strcpy(buf, recover_she_prefix(cmd));
906 45                 if (strlen(buf) > (size_t)10)
907 46                     {
908 47                         sprintf(holdbuf, "( %s %s ) ", buf, cmd_stuff[3]);
909 48                         cmd_stuff[3] = holdbuf;
910 49                     }
911 50                 rcmd_fd = ebr_direct(cmd,
912 51                                     &cmd_stuff[0], cpy->ap, shalcp_port, cpy,
913 52                                     cmd_stuff[1], cmd_stuff[2],
914 53                                     cmd_stuff[3],
915 54                                     );
916 55             }
917 56         if (debugmode)
918 57             {
919 58                 rpe_log_statel(
920 59                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
921 60                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
922 61             }
923 62         if (rcmd_fd == NULL)
924 63             {
925 64                 pto_failcode = 2;
926 65                 pto_error = "cannot set up remote connection";
927 66                 goto send_0;
928 67             }
929 68         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
930 69             "telnet", "telnet"))
931 70             {
932 71                 /* variable for dealing with sym path and cross restore */
933 72                 char path[CDL_HOST_LEN*10];
934 73                 char *symname; /* for returned array of STG names */
935 74                 int numgroups = 0;
936 75                 int addindex = 0;
937 76                 int rc = 0;
938 77                 boolsym_by_symnameok = FALSE;
939 78                 /* fresh connection method here
940 79                  */
941 80                 /*
942 81                  * Set up the rcmd connection (ssh method) to the client.
943 82                  */
944 83                 if (! cpy->ap, shalcp_port)
945 84                     {
946 85                         struct servent *sp;
947 86                         /*
948 87                          * Try to get the port number again.
949 88                          */
949 89                         if (isp == getservbyname("shell", "tcp")) != NULL)
950 90                             {
951 91                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
952 92                                 cpy->ap, shalcp_port = 1;
953 93                             }
954 94                         else
955 95                             {
956 96                                 pto_failcode = 1;
957 97                                 pto_error = "shell/tcp service not found";
958 98                                 goto send_0;
959 99                             }
960 100                     }
961 101                 /*
962 102                  * check for SSL enabled work item and if present add
963 103                  * SSL information to command string as an environment
964 104                  * variable--but first check for cross restore and make
965 105                  * appropriate adjustments
966 106                  */
967 107             }
968 108         if (debugmode)
969 109             {
970 110                 rpe_log_statel(
971 111                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
972 112                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
973 113             }
974 114         if (rcmd_fd == NULL)
975 115             {
976 116                 pto_failcode = 2;
977 117                 pto_error = "cannot set up remote connection";
978 118                 goto send_0;
979 119             }
980 120         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
981 121             "telnet", "telnet"))
982 122             {
983 123                 /* variable for dealing with sym path and cross restore */
984 124                 char path[CDL_HOST_LEN*10];
985 125                 char *symname; /* for returned array of STG names */
986 126                 int numgroups = 0;
987 127                 int addindex = 0;
988 128                 int rc = 0;
989 129                 boolsym_by_symnameok = FALSE;
990 130                 /* fresh connection method here
991 131                  */
992 132                 /*
993 133                  * Set up the rcmd connection (ssh method) to the client.
994 134                  */
995 135                 if (! cpy->ap, shalcp_port)
996 136                     {
997 137                         struct servent *sp;
998 138                         /*
999 139                          * Try to get the port number again.
1000 140                          */
1001 141                         if (isp == getservbyname("shell", "tcp")) != NULL)
1002 142                             {
1003 143                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1004 144                                 cpy->ap, shalcp_port = 1;
1005 145                             }
1006 146                         else
1007 147                             {
1008 148                                 pto_failcode = 1;
1009 149                                 pto_error = "shell/tcp service not found";
1010 150                                 goto send_0;
1011 151                             }
1012 152                     }
1013 153                 /*
1014 154                  * check for SSL enabled work item and if present add
1015 155                  * SSL information to command string as an environment
1016 156                  * variable--but first check for cross restore and make
1017 157                  * appropriate adjustments
1018 158                  */
1019 159             }
1020 160         if (debugmode)
1021 161             {
1022 162                 rpe_log_statel(
1023 163                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1024 164                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1025 165             }
1026 166         if (rcmd_fd == NULL)
1027 167             {
1028 168                 pto_failcode = 2;
1029 169                 pto_error = "cannot set up remote connection";
1030 170                 goto send_0;
1031 171             }
1032 172         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1033 173             "telnet", "telnet"))
1034 174             {
1035 175                 /* variable for dealing with sym path and cross restore */
1036 176                 char path[CDL_HOST_LEN*10];
1037 177                 char *symname; /* for returned array of STG names */
1038 178                 int numgroups = 0;
1039 179                 int addindex = 0;
1040 180                 int rc = 0;
1041 181                 boolsym_by_symnameok = FALSE;
1042 182                 /* fresh connection method here
1043 184                  */
1044 185                 /*
1045 186                  * Set up the rcmd connection (ssh method) to the client.
1046 187                  */
1047 188                 if (! cpy->ap, shalcp_port)
1048 189                     {
1049 190                         struct servent *sp;
1050 191                         /*
1051 192                          * Try to get the port number again.
1052 193                          */
1053 194                         if (isp == getservbyname("shell", "tcp")) != NULL)
1054 195                             {
1055 196                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1056 197                                 cpy->ap, shalcp_port = 1;
1057 198                             }
1058 199                         else
1059 200                             {
1060 201                                 pto_failcode = 1;
1061 202                                 pto_error = "shell/tcp service not found";
1062 203                                 goto send_0;
1063 204                             }
1064 205                     }
1065 206                 /*
1066 207                  * check for SSL enabled work item and if present add
1067 208                  * SSL information to command string as an environment
1069 210                  */
1070 211             }
1071 212         if (debugmode)
1072 213             {
1073 214                 rpe_log_statel(
1074 215                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1075 216                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1076 217             }
1077 218         if (rcmd_fd == NULL)
1078 219             {
1079 220                 pto_failcode = 2;
1080 221                 pto_error = "cannot set up remote connection";
1081 222                 goto send_0;
1082 223             }
1083 224         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1084 225             "telnet", "telnet"))
1085 226             {
1086 227                 /* variable for dealing with sym path and cross restore */
1087 228                 char path[CDL_HOST_LEN*10];
1088 229                 char *symname; /* for returned array of STG names */
1089 230                 int numgroups = 0;
1090 231                 int addindex = 0;
1091 232                 int rc = 0;
1092 233                 boolsym_by_symnameok = FALSE;
1093 234                 /* fresh connection method here
1094 235                  */
1095 236                 /*
1096 237                  * Set up the rcmd connection (ssh method) to the client.
1097 238                  */
1098 239                 if (! cpy->ap, shalcp_port)
1099 240                     {
1100 241                         struct servent *sp;
1101 242                         /*
1102 243                          * Try to get the port number again.
1103 244                          */
1104 245                         if (isp == getservbyname("shell", "tcp")) != NULL)
1105 246                             {
1106 247                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1107 248                                 cpy->ap, shalcp_port = 1;
1108 249                             }
1109 250                         else
1110 251                             {
1111 252                                 pto_failcode = 1;
1112 253                                 pto_error = "shell/tcp service not found";
1113 254                                 goto send_0;
1114 255                             }
1115 256                     }
1116 257                 /*
1117 258                  * check for SSL enabled work item and if present add
1118 259                  * SSL information to command string as an environment
1119 260                  * variable--but first check for cross restore and make
1120 261                  * appropriate adjustments
1121 262                  */
1122 263             }
1123 264         if (debugmode)
1124 265             {
1125 266                 rpe_log_statel(
1126 267                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1127 268                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1128 269             }
1129 270         if (rcmd_fd == NULL)
1130 271             {
1131 272                 pto_failcode = 2;
1132 273                 pto_error = "cannot set up remote connection";
1133 274                 goto send_0;
1134 275             }
1135 276         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1136 277             "telnet", "telnet"))
1137 278             {
1138 279                 /* variable for dealing with sym path and cross restore */
1139 280                 char path[CDL_HOST_LEN*10];
1140 281                 char *symname; /* for returned array of STG names */
1141 282                 int numgroups = 0;
1142 283                 int addindex = 0;
1143 284                 int rc = 0;
1144 285                 boolsym_by_symnameok = FALSE;
1145 286                 /* fresh connection method here
1146 287                  */
1147 288                 /*
1148 289                  * Set up the rcmd connection (ssh method) to the client.
1149 290                  */
1150 291                 if (! cpy->ap, shalcp_port)
1151 292                     {
1152 293                         struct servent *sp;
1153 294                         /*
1154 295                          * Try to get the port number again.
1155 296                          */
1156 297                         if (isp == getservbyname("shell", "tcp")) != NULL)
1157 298                             {
1158 299                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1159 300                                 cpy->ap, shalcp_port = 1;
1160 301                             }
1161 302                         else
1162 303                             {
1163 304                                 pto_failcode = 1;
1164 305                                 pto_error = "shell/tcp service not found";
1165 306                                 goto send_0;
1166 307                             }
1167 308                     }
1168 309                 /*
1169 310                  * check for SSL enabled work item and if present add
1170 311                  * SSL information to command string as an environment
1171 312                  * variable--but first check for cross restore and make
1172 313                  * appropriate adjustments
1173 314                  */
1174 315             }
1175 316         if (debugmode)
1176 317             {
1177 318                 rpe_log_statel(
1178 319                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1179 320                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1180 321             }
1181 322         if (rcmd_fd == NULL)
1182 323             {
1183 324                 pto_failcode = 2;
1184 325                 pto_error = "cannot set up remote connection";
1185 326                 goto send_0;
1186 327             }
1187 328         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1188 329             "telnet", "telnet"))
1189 330             {
1190 331                 /* variable for dealing with sym path and cross restore */
1191 332                 char path[CDL_HOST_LEN*10];
1192 333                 char *symname; /* for returned array of STG names */
1193 334                 int numgroups = 0;
1194 335                 int addindex = 0;
1195 336                 int rc = 0;
1196 337                 boolsym_by_symnameok = FALSE;
1197 338                 /* fresh connection method here
1198 339                  */
1199 340                 /*
1200 341                  * Set up the rcmd connection (ssh method) to the client.
1201 342                  */
1202 343                 if (! cpy->ap, shalcp_port)
1203 344                     {
1204 345                         struct servent *sp;
1205 346                         /*
1206 347                          * Try to get the port number again.
1207 348                          */
1208 349                         if (isp == getservbyname("shell", "tcp")) != NULL)
1209 350                             {
1210 351                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1211 352                                 cpy->ap, shalcp_port = 1;
1212 353                             }
1213 354                         else
1214 355                             {
1215 356                                 pto_failcode = 1;
1216 357                                 pto_error = "shell/tcp service not found";
1217 358                                 goto send_0;
1218 359                             }
1219 360                     }
1220 361                 /*
1221 362                  * check for SSL enabled work item and if present add
1222 363                  * SSL information to command string as an environment
1223 364                  * variable--but first check for cross restore and make
1224 365                  * appropriate adjustments
1225 366                  */
1226 367             }
1227 368         if (debugmode)
1228 369             {
1229 370                 rpe_log_statel(
1230 371                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1231 372                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1232 373             }
1233 374         if (rcmd_fd == NULL)
1234 375             {
1235 376                 pto_failcode = 2;
1236 377                 pto_error = "cannot set up remote connection";
1237 378                 goto send_0;
1238 379             }
1239 380         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1240 381             "telnet", "telnet"))
1241 382             {
1242 383                 /* variable for dealing with sym path and cross restore */
1243 384                 char path[CDL_HOST_LEN*10];
1244 385                 char *symname; /* for returned array of STG names */
1245 386                 int numgroups = 0;
1246 387                 int addindex = 0;
1247 388                 int rc = 0;
1248 389                 boolsym_by_symnameok = FALSE;
1249 390                 /* fresh connection method here
1250 391                  */
1251 392                 /*
1252 393                  * Set up the rcmd connection (ssh method) to the client.
1253 394                  */
1254 395                 if (! cpy->ap, shalcp_port)
1255 396                     {
1256 397                         struct servent *sp;
1257 398                         /*
1258 399                          * Try to get the port number again.
1259 400                          */
1260 401                         if (isp == getservbyname("shell", "tcp")) != NULL)
1261 402                             {
1262 403                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1263 404                                 cpy->ap, shalcp_port = 1;
1264 405                             }
1265 406                         else
1266 407                             {
1267 408                                 pto_failcode = 1;
1268 409                                 pto_error = "shell/tcp service not found";
1269 410                                 goto send_0;
1270 411                             }
1271 412                     }
1272 413                 /*
1273 414                  * check for SSL enabled work item and if present add
1274 415                  * SSL information to command string as an environment
1275 416                  * variable--but first check for cross restore and make
1276 417                  * appropriate adjustments
1277 418                  */
1278 419             }
1279 420         if (debugmode)
1280 421             {
1281 422                 rpe_log_statel(
1282 423                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1283 424                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1284 425             }
1285 426         if (rcmd_fd == NULL)
1286 427             {
1287 428                 pto_failcode = 2;
1288 429                 pto_error = "cannot set up remote connection";
1289 430                 goto send_0;
1290 431             }
1291 432         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1292 433             "telnet", "telnet"))
1293 434             {
1294 435                 /* variable for dealing with sym path and cross restore */
1295 436                 char path[CDL_HOST_LEN*10];
1296 437                 char *symname; /* for returned array of STG names */
1297 438                 int numgroups = 0;
1298 439                 int addindex = 0;
1299 440                 int rc = 0;
1300 441                 boolsym_by_symnameok = FALSE;
1301 442                 /* fresh connection method here
1302 443                  */
1303 444                 /*
1304 445                  * Set up the rcmd connection (ssh method) to the client.
1305 446                  */
1306 447                 if (! cpy->ap, shalcp_port)
1307 448                     {
1308 449                         struct servent *sp;
1309 450                         /*
1310 451                          * Try to get the port number again.
1311 452                          */
1312 453                         if (isp == getservbyname("shell", "tcp")) != NULL)
1313 454                             {
1314 455                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1315 456                                 cpy->ap, shalcp_port = 1;
1316 457                             }
1317 458                         else
1318 459                             {
1319 460                                 pto_failcode = 1;
1320 461                                 pto_error = "shell/tcp service not found";
1321 462                                 goto send_0;
1322 463                             }
1323 464                     }
1324 465                 /*
1325 466                  * check for SSL enabled work item and if present add
1326 467                  * SSL information to command string as an environment
1327 468                  * variable--but first check for cross restore and make
1328 469                  * appropriate adjustments
1329 470                  */
1330 471             }
1331 472         if (debugmode)
1332 473             {
1333 474                 rpe_log_statel(
1334 475                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1335 476                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1336 477             }
1337 478         if (rcmd_fd == NULL)
1338 479             {
1339 480                 pto_failcode = 2;
1340 481                 pto_error = "cannot set up remote connection";
1341 482                 goto send_0;
1342 483             }
1343 484         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1344 485             "telnet", "telnet"))
1345 486             {
1346 487                 /* variable for dealing with sym path and cross restore */
1347 488                 char path[CDL_HOST_LEN*10];
1348 489                 char *symname; /* for returned array of STG names */
1349 490                 int numgroups = 0;
1350 491                 int addindex = 0;
1351 492                 int rc = 0;
1352 493                 boolsym_by_symnameok = FALSE;
1353 494                 /* fresh connection method here
1354 495                  */
1355 496                 /*
1356 497                  * Set up the rcmd connection (ssh method) to the client.
1357 498                  */
1358 499                 if (! cpy->ap, shalcp_port)
1359 500                     {
1360 501                         struct servent *sp;
1361 502                         /*
1362 503                          * Try to get the port number again.
1363 504                          */
1364 505                         if (isp == getservbyname("shell", "tcp")) != NULL)
1365 506                             {
1366 507                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1367 508                                 cpy->ap, shalcp_port = 1;
1368 509                             }
1369 510                         else
1370 511                             {
1371 512                                 pto_failcode = 1;
1372 513                                 pto_error = "shell/tcp service not found";
1373 514                                 goto send_0;
1374 515                             }
1375 516                     }
1376 517                 /*
1377 518                  * check for SSL enabled work item and if present add
1378 519                  * SSL information to command string as an environment
1379 520                  * variable--but first check for cross restore and make
1380 521                  * appropriate adjustments
1381 522                  */
1382 523             }
1383 524         if (debugmode)
1384 525             {
1385 526                 rpe_log_statel(
1386 527                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1387 528                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1388 529             }
1389 530         if (rcmd_fd == NULL)
1390 531             {
1391 532                 pto_failcode = 2;
1392 533                 pto_error = "cannot set up remote connection";
1393 534                 goto send_0;
1394 535             }
1395 536         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1396 537             "telnet", "telnet"))
1397 538             {
1398 539                 /* variable for dealing with sym path and cross restore */
1399 540                 char path[CDL_HOST_LEN*10];
1400 541                 char *symname; /* for returned array of STG names */
1401 542                 int numgroups = 0;
1402 543                 int addindex = 0;
1403 544                 int rc = 0;
1404 545                 boolsym_by_symnameok = FALSE;
1405 546                 /* fresh connection method here
1406 547                  */
1407 548                 /*
1408 549                  * Set up the rcmd connection (ssh method) to the client.
1409 550                  */
1410 551                 if (! cpy->ap, shalcp_port)
1411 552                     {
1412 553                         struct servent *sp;
1413 554                         /*
1414 555                          * Try to get the port number again.
1415 556                          */
1416 557                         if (isp == getservbyname("shell", "tcp")) != NULL)
1417 558                             {
1418 559                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1419 560                                 cpy->ap, shalcp_port = 1;
1420 561                             }
1421 562                         else
1422 563                             {
1423 564                                 pto_failcode = 1;
1424 565                                 pto_error = "shell/tcp service not found";
1425 566                                 goto send_0;
1426 567                             }
1427 568                     }
1428 569                 /*
1429 570                  * check for SSL enabled work item and if present add
1430 571                  * SSL information to command string as an environment
1431 572                  * variable--but first check for cross restore and make
1432 573                  * appropriate adjustments
1433 574                  */
1434 575             }
1435 576         if (debugmode)
1436 577             {
1437 578                 rpe_log_statel(
1438 579                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1439 580                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1440 581             }
1441 582         if (rcmd_fd == NULL)
1442 583             {
1443 584                 pto_failcode = 2;
1444 585                 pto_error = "cannot set up remote connection";
1445 586                 goto send_0;
1446 587             }
1447 588         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1448 589             "telnet", "telnet"))
1449 590             {
1450 591                 /* variable for dealing with sym path and cross restore */
1451 592                 char path[CDL_HOST_LEN*10];
1452 593                 char *symname; /* for returned array of STG names */
1453 594                 int numgroups = 0;
1454 595                 int addindex = 0;
1455 596                 int rc = 0;
1456 597                 boolsym_by_symnameok = FALSE;
1457 598                 /* fresh connection method here
1458 599                  */
1459 600                 /*
1460 601                  * Set up the rcmd connection (ssh method) to the client.
1461 602                  */
1462 603                 if (! cpy->ap, shalcp_port)
1463 604                     {
1464 605                         struct servent *sp;
1465 606                         /*
1466 607                          * Try to get the port number again.
1467 608                          */
1468 609                         if (isp == getservbyname("shell", "tcp")) != NULL)
1469 610                             {
1470 611                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1471 612                                 cpy->ap, shalcp_port = 1;
1472 613                             }
1473 614                         else
1474 615                             {
1475 616                                 pto_failcode = 1;
1476 617                                 pto_error = "shell/tcp service not found";
1477 618                                 goto send_0;
1478 619                             }
1479 620                     }
1480 621                 /*
1481 622                  * check for SSL enabled work item and if present add
1482 623                  * SSL information to command string as an environment
1483 624                  * variable--but first check for cross restore and make
1484 625                  * appropriate adjustments
1485 626                  */
1486 627             }
1487 628         if (debugmode)
1488 629             {
1489 630                 rpe_log_statel(
1490 631                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1491 632                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1492 633             }
1493 634         if (rcmd_fd == NULL)
1494 635             {
1495 636                 pto_failcode = 2;
1496 637                 pto_error = "cannot set up remote connection";
1497 638                 goto send_0;
1498 639             }
1499 640         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1500 641             "telnet", "telnet"))
1501 642             {
1502 643                 /* variable for dealing with sym path and cross restore */
1503 644                 char path[CDL_HOST_LEN*10];
1504 645                 char *symname; /* for returned array of STG names */
1505 646                 int numgroups = 0;
1506 647                 int addindex = 0;
1507 648                 int rc = 0;
1508 649                 boolsym_by_symnameok = FALSE;
1509 650                 /* fresh connection method here
1510 651                  */
1511 652                 /*
1512 653                  * Set up the rcmd connection (ssh method) to the client.
1513 654                  */
1514 655                 if (! cpy->ap, shalcp_port)
1515 656                     {
1516 657                         struct servent *sp;
1517 658                         /*
1518 659                          * Try to get the port number again.
1519 660                          */
1520 661                         if (isp == getservbyname("shell", "tcp")) != NULL)
1521 662                             {
1522 663                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1523 664                                 cpy->ap, shalcp_port = 1;
1524 665                             }
1525 666                         else
1526 667                             {
1527 668                                 pto_failcode = 1;
1528 669                                 pto_error = "shell/tcp service not found";
1529 670                                 goto send_0;
1530 671                             }
1531 672                     }
1532 673                 /*
1533 674                  * check for SSL enabled work item and if present add
1534 675                  * SSL information to command string as an environment
1535 676                  * variable--but first check for cross restore and make
1536 677                  * appropriate adjustments
1537 678                  */
1538 679             }
1539 680         if (debugmode)
1540 681             {
1541 682                 rpe_log_statel(
1542 683                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1543 684                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1544 685             }
1545 686         if (rcmd_fd == NULL)
1546 687             {
1547 688                 pto_failcode = 2;
1548 689                 pto_error = "cannot set up remote connection";
1549 690                 goto send_0;
1550 691             }
1551 692         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1552 693             "telnet", "telnet"))
1553 694             {
1554 695                 /* variable for dealing with sym path and cross restore */
1555 696                 char path[CDL_HOST_LEN*10];
1556 697                 char *symname; /* for returned array of STG names */
1557 698                 int numgroups = 0;
1558 699                 int addindex = 0;
1559 700                 int rc = 0;
1560 701                 boolsym_by_symnameok = FALSE;
1561 702                 /* fresh connection method here
1562 703                  */
1563 704                 /*
1564 705                  * Set up the rcmd connection (ssh method) to the client.
1565 706                  */
1566 707                 if (! cpy->ap, shalcp_port)
1567 708                     {
1568 709                         struct servent *sp;
1569 710                         /*
1570 711                          * Try to get the port number again.
1571 712                          */
1572 713                         if (isp == getservbyname("shell", "tcp")) != NULL)
1573 714                             {
1574 715                                 cpy->ap, shalcp_port = (ushort_t)sp->port;
1575 716                                 cpy->ap, shalcp_port = 1;
1576 717                             }
1577 718                         else
1578 719                             {
1579 720                                 pto_failcode = 1;
1580 721                                 pto_error = "shell/tcp service not found";
1581 722                                 goto send_0;
1582 723                             }
1583 724                     }
1584 725                 /*
1585 726                  * check for SSL enabled work item and if present add
1586 727                  * SSL information to command string as an environment
1587 728                  * variable--but first check for cross restore and make
1588 729                  * appropriate adjustments
1589 730                  */
1590 731             }
1591 732         if (debugmode)
1592 733             {
1593 734                 rpe_log_statel(
1594 735                     0, "RMXPROC: rcmd returned fd %d, stderr fd %d\n",
1595 736                     NULL != rcmd_fd ? rcmd_fd[0] : -1, rcmd_stderr);
1596 737             }
1597 738         if (rcmd_fd == NULL)
1598 739             {
1599 740                 pto_failcode = 2;
1600 741                 pto_error = "cannot set up remote connection";
1601 742                 goto send_0;
1602 743             }
1603 744         else if (0 == strcmp(method, "ssh")) || (0 == strcmp(
1604 745            
```



```

1329 4      char *p4;
1330 4      while (i, == *p3)
1331 4      {
1332 4          *p3++; /* advance to start of next word */
1333 4      }
1334 4      p4 = p3; /* save this as start of the prefix */
1335 4      for (i = 0; i = *p3; i++)
1336 4      {
1337 4          if (i == *p3 && (i == p3[1]) ||
1338 4              (i == *p3 && (i == p3[1])))
1339 4          {
1340 4              break;
1341 4          }
1342 4          if (*p3 == 0) /* if found the prefix */
1343 4          {
1344 4              char *tempChar = NULL;
1345 4              strcpy(tempChar, p4, p3-p4);
1346 4              strcat(tempChar, "\0");
1347 4              if (i == *p3)
1348 4              {
1349 4                  if (i == p3[1])
1350 4                  {
1351 4                      *p3++;
1352 4                      tempChar = strchr(p3, '\n');
1353 4                      if (NULL != tempChar)
1354 4                      {
1355 4                          tempChar[0] = ' ';
1356 4                      }
1357 4                      strcpy(p4, p3+1);
1358 4                  }
1359 4              }
1360 4              /*
1361 4               * now, if it's a network cross recovery, get password.
1362 4               * destination target -- don't use source target from
1363 4               * psw pointer to work from config struct backup was done
1364 4               */
1365 4              if (0 != strcmp(
1366 4                  rcmd_stuff[0], psw->username)) /* if x-recovery */
1367 4              {
1368 4                  RBC_WORKITEM *pwiz = psw;
1369 4              }
1370 4              for (pwg = cxd->ed_config->pgrouplist; NULL != pwg;
1371 4                  pwg = pwg->next)
1372 4              {
1373 4                  for (pwi = pwg->pwlist; NULL != pwi; pwi = pwi->next)
1374 4                  {
1375 4                      if (strcmp(pwi->username, rcmd_stuff[0]))
1376 4                      {
1377 4                          goto gotit2;
1378 4                      }
1379 4                  }
1380 4              }
1381 4              gotit2:
1382 4          }
1383 4      }
1384 4      return 0;
1385 4  }
1386 4  }
1387 4  }
1388 4  }
1389 4  }
1390 4  }

```

Mon Oct 13 16:00:37 2008

RSLAUMAIN.25

Page 91 of 134

```

1391 4      /*
1392 4      * if we did not find a wi for the target system,
1393 4      * put user prompting here.
1394 4      */
1395 4      if (NULL == pwi)
1396 4      {
1397 4          pwi = pwiz; /* restore old value if i found */
1398 4      }
1399 4      /*
1400 4      * check for an encrypted password
1401 4      */
1402 4      if (NULL != pwi && (
1403 4          pwi->flags & WORKITEM_FLAGS_ENCRYPTED_PASSWORD))
1404 4      {
1405 4          printf(
1406 4              "buf, %s -target %s -tea %s -login %s -password %s", p2,
1407 4              (target != NULL) ? target :
1408 4              "target",
1409 4              (target != NULL) ? target :
1410 4              "target",
1411 4              pwi->cm_cint_target != NULL ? pwi->cm_cint_target : "?",
1412 4              (
1413 4                  (
1414 4                      pwi->username != NULL) ? pwi->username : "?",
1415 4                      (
1416 4                          pwi->password != NULL) ? pwi->password : "?");
1417 4                  )
1418 4              );
1419 4          else
1420 4          {
1421 4              printf(
1422 4                  "buf, %s -target %s -tea %s -login %s -password %s", p2,
1423 4                  (target != NULL) ? target :
1424 4                  "target",
1425 4                  (target != NULL) ? target :
1426 4                  "target",
1427 4                  pwi->cm_cint_target != NULL ? pwi->cm_cint_target : "?",
1428 4                  (
1429 4                      (
1430 4                          pwi->username != NULL) ? pwi->username : "?",
1431 4                          (
1432 4                              pwi->password != NULL) ? pwi->password : "?");
1433 4                      )
1434 4                  );
1435 4              if (NULL != pwi && (CNCN_RSH != pwi->connection_type))
1436 4              {
1437 4                  if (CNCN_NETWORK != pwi->connection_type)
1438 4                  {
1439 4                      if (
1440 4                          !strcmp(rcmd_stuff[0], "X-PROG fd,
1441 4                              EXIMPROXMSG,ADPROC,WARNING, 0,
1442 4                              "work item %s" specified
1443 4                              "connection method to but client
1444 4                              "was installed to use the network
1445 4                              "method -- using network method",
1446 4                              pwi->name,
1447 4                              CNCN_RSH == pwi->connection_type) ? "Rsh" :
1448 4                              (
1449 4                                  (
1450 4                                      CNCN_EMULATOR == pwi->EdmLink :
1451 4                                      (
1452 4                                          CNCN_DIRECT == (
1453 4                                              pwi->connection_type) ? "Direct" :
1454 4                                              CNCN_SOCKET == pwi->connection_type) ? "Socket" :
1455 4                                              CNCN_NETWORK == pwi->connection_type) ? "Network" :
1456 4                                              "???");
1457 4                                      )
1458 4                                  )
1459 4                              )
1460 4                              )
1461 4                              )
1462 4                              )
1463 4                              )
1464 4                              )
1465 4                              )
1466 4                              )
1467 4                              )
1468 4                              )
1469 4                              )
1470 4                              )
1471 4                              )
1472 4                              )
1473 4                              )
1474 4                              )
1475 4                              )
1476 4                              )
1477 4                              )
1478 4                              )
1479 4                              )
1480 4                              )
1481 4                              )
1482 4                              )
1483 4                              )
1484 4                              )
1485 4                              )
1486 4                              )
1487 4                              )
1488 4                              )
1489 4                              )
1490 4                              )
1491 4                              )
1492 4                              )
1493 4                              )
1494 4                              )
1495 4                              )
1496 4                              )
1497 4                              )
1498 4                              )
1499 4                              )
1500 4                              )
1501 4                              )
1502 4                              )
1503 4                              )
1504 4                              )
1505 4                              )
1506 4                              )
1507 4                              )
1508 4                              )
1509 4                              )
1510 4                              )
1511 4                              )
1512 4                              )
1513 4                              )
1514 4                              )
1515 4                              )
1516 4                              )
1517 4                              )
1518 4                              )
1519 4                              )
1520 4                              )
1521 4                              )
1522 4                              )
1523 4                              )
1524 4                              )
1525 4                              )
1526 4                              )
1527 4                              )
1528 4                              )
1529 4                              )
1530 4                              )
1531 4                              )
1532 4                              )
1533 4                              )
1534 4                              )
1535 4                              )
1536 4                              )
1537 4                              )
1538 4                              )
1539 4                              )
1540 4                              )
1541 4                              )
1542 4                              )
1543 4                              )
1544 4                              )
1545 4                              )
1546 4                              )
1547 4                              )
1548 4                              )
1549 4                              )
1550 4                              )
1551 4                              )
1552 4                              )
1553 4                              )
1554 4                              )
1555 4                              )
1556 4                              )
1557 4                              )
1558 4                              )
1559 4                              )
1560 4                              )
1561 4                              )
1562 4                              )
1563 4                              )
1564 4                              )
1565 4                              )
1566 4                              )
1567 4                              )
1568 4                              )
1569 4                              )
1570 4                              )
1571 4                              )
1572 4                              )
1573 4                              )
1574 4                              )
1575 4                              )
1576 4                              )
1577 4                              )
1578 4                              )
1579 4                              )
1580 4                              )
1581 4                              )
1582 4                              )
1583 4                              )
1584 4                              )
1585 4                              )
1586 4                              )
1587 4                              )
1588 4                              )
1589 4                              )
1590 4                              )
1591 4                              )
1592 4                              )
1593 4                              )
1594 4                              )
1595 4                              )
1596 4                              )
1597 4                              )
1598 4                              )
1599 4                              )
1600 4                              )
1601 4                              )
1602 4                              )
1603 4                              )
1604 4                              )
1605 4                              )
1606 4                              )
1607 4                              )
1608 4                              )
1609 4                              )
1610 4                              )
1611 4                              )
1612 4                              )
1613 4                              )
1614 4                              )
1615 4                              )
1616 4                              )
1617 4                              )
1618 4                              )
1619 4                              )
1620 4                              )
1621 4                              )
1622 4                              )
1623 4                              )
1624 4                              )
1625 4                              )
1626 4                              )
1627 4                              )
1628 4                              )
1629 4                              )
1630 4                              )
1631 4                              )
1632 4                              )
1633 4                              )
1634 4                              )
1635 4                              )
1636 4                              )
1637 4                              )
1638 4                              )
1639 4                              )
1640 4                              )
1641 4                              )
1642 4                              )
1643 4                              )
1644 4                              )
1645 4                              )
1646 4                              )
1647 4                              )
1648 4                              )
1649 4                              )
1650 4                              )
1651 4                              )
1652 4                              )
1653 4                              )
1654 4                              )
1655 4                              )
1656 4                              )
1657 4                              )
1658 4                              )
1659 4                              )
1660 4                              )
1661 4                              )
1662 4                              )
1663 4                              )
1664 4                              )
1665 4                              )
1666 4                              )
1667 4                              )
1668 4                              )
1669 4                              )
1670 4                              )
1671 4                              )
1672 4                              )
1673 4                              )
1674 4                              )
1675 4                              )
1676 4                              )
1677 4                              )
1678 4                              )
1679 4                              )
1680 4                              )
1681 4                              )
1682 4                              )
1683 4                              )
1684 4                              )
1685 4                              )
1686 4                              )
1687 4                              )
1688 4                              )
1689 4                              )
1690 4                              )
1691 4                              )
1692 4                              )
1693 4                              )
1694 4                              )
1695 4                              )
1696 4                              )
1697 4                              )
1698 4                              )
1699 4                              )
1700 4                              )
1701 4                              )
1702 4                              )
1703 4                              )
1704 4                              )
1705 4                              )
1706 4                              )
1707 4                              )
1708 4                              )
1709 4                              )
1710 4                              )
1711 4                              )
1712 4                              )
1713 4                              )
1714 4                              )
1715 4                              )
1716 4                              )
1717 4                              )
1718 4                              )
1719 4                              )
1720 4                              )
1721 4                              )
1722 4                              )
1723 4                              )
1724 4                              )
1725 4                              )
1726 4                              )
1727 4                              )
1728 4                              )
1729 4                              )
1730 4                              )
1731 4                              )
1732 4                              )
1733 4                              )
1734 4                              )
1735 4                              )
1736 4                              )
1737 4                              )
1738 4                              )
1739 4                              )
1740 4                              )
1741 4                              )
1742 4                              )
1743 4                              )
1744 4                              )
1745 4                              )
1746 4                              )
1747 4                              )
1748 4                              )
1749 4                              )
1750 4                              )
1751 4                              )
1752 4                              )
1753 4                              )
1754 4                              )
1755 4                              )
1756 4                              )
1757 4                              )
1758 4                              )
1759 4                              )
1760 4                              )
1761 4                              )
1762 4                              )
1763 4                              )
1764 4                              )
1765 4                              )
1766 4                              )
1767 4                              )
1768 4                              )
1769 4                              )
1770 4                              )
1771 4                              )
1772 4                              )
1773 4                              )
1774 4                              )
1775 4                              )
1776 4                              )
1777 4                              )
1778 4                              )
1779 4                              )
1780 4                              )
1781 4                              )
1782 4                              )
1783 4                              )
1784 4                              )
1785 4                              )
1786 4                              )
1787 4                              )
1788 4                              )
1789 4                              )
1790 4                              )
1791 4                              )
1792 4                              )
1793 4                              )
1794 4                              )
1795 4                              )
1796 4                              )
1797 4                              )
1798 4                              )
1799 4                              )
1800 4                              )
1801 4                              )
1802 4                              )
1803 4                              )
1804 4                              )
1805 4                              )
1806 4                              )
1807 4                              )
1808 4                              )
1809 4                              )
1810 4                              )
1811 4                              )
1812 4                              )
1813 4                              )
1814 4                              )
1815 4                              )
1816 4                              )
1817 4                              )
1818 4                              )
1819 4                              )
1820 4                              )
1821 4                              )
1822 4                              )
1823 4                              )
1824 4                              )
1825 4                              )
1826 4                              )
1827 4                              )
1828 4                              )
1829 4                              )
1830 4                              )
1831 4                              )
1832 4                              )
1833 4                              )
1834 4                              )
1835 4                              )
1836 4                              )
1837 4                              )
1838 4                              )
1839 4                              )
1840 4                              )
1841 4                              )
1842 4                              )
1843 4                              )
1844 4                              )
1845 4                              )
1846 4                              )
1847 4                              )
1848 4                              )
1849 4                              )
1850 4                              )
1851 4                              )
1852 4                              )
1853 4                              )
1854 4                              )
1855 4                              )
1856 4                              )
1857 4                              )
1858 4                              )
1859 4                              )
1860 4                              )
1861 4                              )
1862 4                              )
1863 4                              )
1864 4                              )
1865 4                              )
1866 4                              )
1867 4                              )
1868 4                              )
1869 4                              )
1870 4                              )
1871 4                              )
1872 4                              )
1873 4                              )
1874 4                              )
1875 4                              )
1876 4                              )
1877 4                              )
1878 4                              )
1879 4                              )
1880 4                              )
1881 4                              )
1882 4                              )
1883 4                              )
1884 4                              )
1885 4                              )
1886 4                              )
1887 4                              )
1888 4                              )
1889 4                              )
1890 4                              )
1891 4                              )
1892 4                              )
1893 4                              )
1894 4                              )
1895 4                              )
1896 4                              )
1897 4                              )
1898 4                              )
1899 4                              )
1900 4                              )
1901 4                              )
1902 4                              )
1903 4                              )
1904 4                              )
1905 4                              )
1906 4                              )
1907 4                              )
1908 4                              )
1909 4                              )
1910 4                              )
1911 4                              )
1912 4                              )
1913 4                              )
1914 4                              )
1915 4                              )
1916 4                              )
1917 4                              )
1918 4                              )
1919 4                              )
1920 4                              )
1921 4                              )
1922 4                              )
1923 4                              )
1924 4                              )
1925 4                              )
1926 4                              )
1927 4                              )
1928 4                              )
1929 4                              )
1930 4                              )
1931 4                              )
1932 4                              )
1933 4                              )
1934 4                              )
1935 4                              )
1936 4                              )
1937 4                              )
1938 4                              )
1939 4                              )
1940 4                              )
1941 4                              )
1942 4                              )
1943 4                              )
1944 4                              )
1945 4                              )
1946 4                              )
1947 4                              )
1948 4                              )
1949 4                              )
1950 4                              )
1951 4                              )
1952 4                              )
1953 4                              )
1954 4                              )
1955 4                              )
1956 4                              )
1957 4                              )
1958 4                              )
1959 4                              )
1960 4                              )
1961 4                              )
1962 4                              )
1963 4                              )
1964 4                              )
1965 4                              )
1966 4                              )
1967 4                              )
1968 4                              )
1969 4                              )
1970 4                              )
1971 4                              )
1972 4                              )
1973 4                              )
1974 4                              )
1975 4                              )
1976 4                              )
1977 4                              )
1978 4                              )
1979 4                              )
1980 4                              )
1981 4                              )
1982 4                              )
1983 4                              )
1984 4                              )
1985 4                              )
1986 4                              )
1987 4                              )
1988 4                              )
1989 4                              )
1990 4                              )
1991 4                              )
1992 4                              )
1993 4                              )
1994 4                              )
1995 4                              )
1996 4                              )
1997 4                              )
1998 4                              )
1999 4                              )
2000 4                              )

```

Mon Oct 13 16:00:37 2008

RSLAUMAIN.26

Page 92 of 134


```

1553 4         }
1554 4         socket_port = pwi->connection_port;
1555 4     }
1556 2     if (debugmode)
1557 2     {
1558 2         rbe_log_stats(
1559 2             0, "host= %s port=%d seq=%d\n", socket_host, socket_port, dseqno);
1560 2     }
1561 2     cmd_fd = cmd_fd; /* set valid pointer */
1562 2     /* We need to fix this up...
1563 2     of socket (STR) */
1564 2     Modify sopen to call the right type
1565 2     cmd_fd(0) = sopen(socket_host, 'C', socket_port);
1566 2     if (cmd_fd(0) < 0)
1567 2     {
1568 2         rbe_log_stats(0, "Socket Open error: %d\n", cmd_fd(0));
1569 2         _exit(12);
1570 2     }
1571 2     /*
1572 2     * set up ONLY remote command file descriptors to talk to client
1573 2     * via socket connection. DO NOT overwrite filter_cmd fd's
1574 2     */
1575 2     cmd_fd(1) = dup(cmd_fd(0));
1576 2     /* If we are connected,
1577 2     determine if the user wants normal sockets
1578 2     * or ssl.
1579 2     */
1580 2     if (cmd_fd(0) != -1)
1581 2     {
1582 2         if (pwi->ssl_groupname)
1583 2         {
1584 2             /* send OK status to client waiting on data socket
1585 2             before
1586 2             * data stream
1587 2             */
1588 2             sprintf(buf, "OK: %d socket connection made;
1589 2             (void)write (cmd_fd(0), buf, (int)strlen(buf));
1590 2         }
1591 2     }
1592 2     else /* ssl connect */
1593 2     {
1594 2         char sslname[CDL_MAX_LEN+1];
1595 2         char ssgroup[CDL_MAX_LEN+1];
1596 2         char *argnames; /* for returned array of STG names */
1597 2         int numgroups = 0;
1598 2         int oddindex = 0;
1599 2         int rc = 0;
1600 2         /* First test to see if this is a cross restore. If
1601 2         * so we can't just use the STG group from the source
1602 2         client
1603 2         */
1604 2         if (pwi->ssl_groupname)
1605 2         {
1606 2             memcpy(sslname, pwi->ssl_groupname, CDL_MAX_LEN+1);
1607 2             memcpy(ssgroup, pwi->ssl_groupname, CDL_MAX_LEN+1);
1608 2             argnames = sslname;
1609 2             numgroups = 1;
1610 2         }
1611 2         else
1612 2         {
1613 2             /* Grab the STG entries for this client */
1614 2             rc = CDL_getavailablegroups(
1615 2                 socket_host, pwi->servername);
1616 2             if (rc < 0)
1617 2             {
1618 2                 /* Fail back to network.
1619 2                 If no STG group is available we will
1620 2                 * fail back to network.
1621 2                 */
1622 2                 if (rc == SOCKET_ERROR)
1623 2                 {
1624 2                     /* We need to use Symbath for cross restore to host &= using
1625 2                     errstr, "Unable to use Symbath for cross restore to host &= using
1626 2                     the user's error(0, errstr);
1627 2                     /* Default to network and send normal
1628 2                     handshake.
1629 2                     * send OK status to client waiting on data
1630 2                     socket before
1631 2                     * data stream
1632 2                     */
1633 2                     sprintf(buf, "OK: %d socket connection made;
1634 2                     (void)write (cmd_fd(0), buf, (int)strlen(buf));
1635 2                 }
1636 2                 else
1637 2                 {
1638 2                     /* The destination client is SP enabled!
1639 2                     * to see if the same STG group exists.
1640 2                     If we don't
1641 2                     * find the same one we will use the first,
1642 2                     so set
1643 2                     * that as a default
1644 2                     */
1645 2                     memcpy(sslname, 0, CDL_MAX_LEN+1);
1646 2                     memcpy(ssgroup, 0, CDL_MAX_LEN+1);
1647 2                     argnames = sslname;
1648 2                     numgroups = 1;
1649 2                 }
1650 2             }
1651 2             /* Initialize the index */
1652 2             while (--addindex >= 0)
1653 2             {
1654 2                 if (0 == strcmp(
1655 2                     pwi->ssl_groupname, argnames[addindex]))
1656 2                 {
1657 2                     strategy[addindex] = CDL_MAX_LEN+1;
1658 2                     ssgroup[addindex] = CDL_MAX_LEN+1;
1659 2                     continue;
1660 2                 }
1661 2             }
1662 2             /* We need to get a short name SSL alias if
1663 2             the destination
1664 2             */
1665 2         }
1666 2     }
1667 2 }
1668 2 }
1669 2 }
1670 2 }
1671 2 }
1672 2 }
1673 2 }
1674 2 }
1675 2 }
1676 2 }
1677 2 }
1678 2 }
1679 2 }
1680 2 }
1681 2 }
1682 2 }
1683 2 }
1684 2 }
1685 2 }
1686 2 }
1687 2 }
1688 2 }
1689 2 }
1690 2 }
1691 2 }
1692 2 }
1693 2 }
1694 2 }
1695 2 }
1696 2 }
1697 2 }
1698 2 }
1699 2 }
1700 2 }
1701 2 }
1702 2 }
1703 2 }
1704 2 }
1705 2 }
1706 2 }
1707 2 }
1708 2 }
1709 2 }
1710 2 }
1711 2 }
1712 2 }
1713 2 }
1714 2 }
1715 2 }
1716 2 }
1717 2 }
1718 2 }
1719 2 }
1720 2 }
1721 2 }
1722 2 }
1723 2 }
1724 2 }
1725 2 }
1726 2 }
1727 2 }
1728 2 }
1729 2 }
1730 2 }
1731 2 }
1732 2 }
1733 2 }
1734 2 }
1735 2 }
1736 2 }
1737 2 }
1738 2 }
1739 2 }
1740 2 }
1741 2 }
1742 2 }
1743 2 }
1744 2 }
1745 2 }
1746 2 }
1747 2 }
1748 2 }
1749 2 }
1750 2 }
1751 2 }
1752 2 }
1753 2 }
1754 2 }
1755 2 }
1756 2 }
1757 2 }
1758 2 }
1759 2 }
1760 2 }
1761 2 }
1762 2 }
1763 2 }
1764 2 }
1765 2 }
1766 2 }
1767 2 }
1768 2 }
1769 2 }
1770 2 }
1771 2 }
1772 2 }
1773 2 }
1774 2 }
1775 2 }
1776 2 }
1777 2 }
1778 2 }
1779 2 }
1780 2 }
1781 2 }
1782 2 }
1783 2 }
1784 2 }
1785 2 }
1786 2 }
1787 2 }
1788 2 }
1789 2 }
1790 2 }
1791 2 }
1792 2 }
1793 2 }
1794 2 }
1795 2 }
1796 2 }
1797 2 }
1798 2 }
1799 2 }
1800 2 }
1801 2 }
1802 2 }
1803 2 }
1804 2 }
1805 2 }
1806 2 }
1807 2 }
1808 2 }
1809 2 }
1810 2 }
1811 2 }
1812 2 }
1813 2 }
1814 2 }
1815 2 }
1816 2 }
1817 2 }
1818 2 }
1819 2 }
1820 2 }
1821 2 }
1822 2 }
1823 2 }
1824 2 }
1825 2 }
1826 2 }
1827 2 }
1828 2 }
1829 2 }
1830 2 }
1831 2 }
1832 2 }
1833 2 }
1834 2 }
1835 2 }
1836 2 }
1837 2 }
1838 2 }
1839 2 }
1840 2 }
1841 2 }
1842 2 }
1843 2 }
1844 2 }
1845 2 }
1846 2 }
1847 2 }
1848 2 }
1849 2 }
1850 2 }
1851 2 }
1852 2 }
1853 2 }
1854 2 }
1855 2 }
1856 2 }
1857 2 }
1858 2 }
1859 2 }
1860 2 }
1861 2 }
1862 2 }
1863 2 }
1864 2 }
1865 2 }
1866 2 }
1867 2 }
1868 2 }
1869 2 }
1870 2 }
1871 2 }
1872 2 }
1873 2 }
1874 2 }
1875 2 }
1876 2 }
1877 2 }
1878 2 }
1879 2 }
1880 2 }
1881 2 }
1882 2 }
1883 2 }
1884 2 }
1885 2 }
1886 2 }
1887 2 }
1888 2 }
1889 2 }
1890 2 }
1891 2 }
1892 2 }
1893 2 }
1894 2 }
1895 2 }
1896 2 }
1897 2 }
1898 2 }
1899 2 }
1900 2 }
1901 2 }
1902 2 }
1903 2 }
1904 2 }
1905 2 }
1906 2 }
1907 2 }
1908 2 }
1909 2 }
1910 2 }
1911 2 }
1912 2 }
1913 2 }
1914 2 }
1915 2 }
1916 2 }
1917 2 }
1918 2 }
1919 2 }
1920 2 }
1921 2 }
1922 2 }
1923 2 }
1924 2 }
1925 2 }
1926 2 }
1927 2 }
1928 2 }
1929 2 }
1930 2 }
1931 2 }
1932 2 }
1933 2 }
1934 2 }
1935 2 }
1936 2 }
1937 2 }
1938 2 }
1939 2 }
1940 2 }
1941 2 }
1942 2 }
1943 2 }
1944 2 }
1945 2 }
1946 2 }
1947 2 }
1948 2 }
1949 2 }
1950 2 }
1951 2 }
1952 2 }
1953 2 }
1954 2 }
1955 2 }
1956 2 }
1957 2 }
1958 2 }
1959 2 }
1960 2 }
1961 2 }
1962 2 }
1963 2 }
1964 2 }
1965 2 }
1966 2 }
1967 2 }
1968 2 }
1969 2 }
1970 2 }
1971 2 }
1972 2 }
1973 2 }
1974 2 }
1975 2 }
1976 2 }
1977 2 }
1978 2 }
1979 2 }
1980 2 }
1981 2 }
1982 2 }
1983 2 }
1984 2 }
1985 2 }
1986 2 }
1987 2 }
1988 2 }
1989 2 }
1990 2 }
1991 2 }
1992 2 }
1993 2 }
1994 2 }
1995 2 }
1996 2 }
1997 2 }
1998 2 }
1999 2 }
2000 2 }

```

```

1609 4         * Since it may not exist on the destination client.
1610 4         * We need to see if the destination is SP enabled and
1611 4         * get a good STG group for this. We'll use the same group
1612 4         * if it is available.
1613 4         * If no STG group is available we will
1614 4         * fail back to network.
1615 4         */
1616 4         if (rc == SOCKET_ERROR)
1617 4         {
1618 4             /* We need to use Symbath for cross restore to host &= using
1619 4             errstr, "Unable to use Symbath for cross restore to host &= using
1620 4             the user's error(0, errstr);
1621 4             /* Default to network and send normal
1622 4             handshake.
1623 4             * send OK status to client waiting on data
1624 4             socket before
1625 4             * data stream
1626 4             */
1627 4             sprintf(buf, "OK: %d socket connection made;
1628 4             (void)write (cmd_fd(0), buf, (int)strlen(buf));
1629 4         }
1630 4         else
1631 4         {
1632 4             /* The destination client is SP enabled!
1633 4             * to see if the same STG group exists.
1634 4             If we don't
1635 4             * find the same one we will use the first,
1636 4             so set
1637 4             * that as a default
1638 4             */
1639 4             memcpy(sslname, 0, CDL_MAX_LEN+1);
1640 4             memcpy(ssgroup, 0, CDL_MAX_LEN+1);
1641 4             argnames = sslname;
1642 4             numgroups = 1;
1643 4         }
1644 4         /* Initialize the index */
1645 4         while (--addindex >= 0)
1646 4         {
1647 4             if (0 == strcmp(
1648 4                 pwi->ssl_groupname, argnames[addindex]))
1649 4             {
1650 4                 strategy[addindex] = CDL_MAX_LEN+1;
1651 4                 ssgroup[addindex] = CDL_MAX_LEN+1;
1652 4                 continue;
1653 4             }
1654 4         }
1655 4         /* We need to get a short name SSL alias if
1656 4         the destination
1657 4         */
1658 4     }
1659 4 }
1660 4 }
1661 4 }
1662 4 }
1663 4 }
1664 4 }
1665 4 }
1666 4 }
1667 4 }
1668 4 }
1669 4 }
1670 4 }
1671 4 }
1672 4 }
1673 4 }
1674 4 }
1675 4 }
1676 4 }
1677 4 }
1678 4 }
1679 4 }
1680 4 }
1681 4 }
1682 4 }
1683 4 }
1684 4 }
1685 4 }
1686 4 }
1687 4 }
1688 4 }
1689 4 }
1690 4 }
1691 4 }
1692 4 }
1693 4 }
1694 4 }
1695 4 }
1696 4 }
1697 4 }
1698 4 }
1699 4 }
1700 4 }
1701 4 }
1702 4 }
1703 4 }
1704 4 }
1705 4 }
1706 4 }
1707 4 }
1708 4 }
1709 4 }
1710 4 }
1711 4 }
1712 4 }
1713 4 }
1714 4 }
1715 4 }
1716 4 }
1717 4 }
1718 4 }
1719 4 }
1720 4 }
1721 4 }
1722 4 }
1723 4 }
1724 4 }
1725 4 }
1726 4 }
1727 4 }
1728 4 }
1729 4 }
1730 4 }
1731 4 }
1732 4 }
1733 4 }
1734 4 }
1735 4 }
1736 4 }
1737 4 }
1738 4 }
1739 4 }
1740 4 }
1741 4 }
1742 4 }
1743 4 }
1744 4 }
1745 4 }
1746 4 }
1747 4 }
1748 4 }
1749 4 }
1750 4 }
1751 4 }
1752 4 }
1753 4 }
1754 4 }
1755 4 }
1756 4 }
1757 4 }
1758 4 }
1759 4 }
1760 4 }
1761 4 }
1762 4 }
1763 4 }
1764 4 }
1765 4 }
1766 4 }
1767 4 }
1768 4 }
1769 4 }
1770 4 }
1771 4 }
1772 4 }
1773 4 }
1774 4 }
1775 4 }
1776 4 }
1777 4 }
1778 4 }
1779 4 }
1780 4 }
1781 4 }
1782 4 }
1783 4 }
1784 4 }
1785 4 }
1786 4 }
1787 4 }
1788 4 }
1789 4 }
1790 4 }
1791 4 }
1792 4 }
1793 4 }
1794 4 }
1795 4 }
1796 4 }
1797 4 }
1798 4 }
1799 4 }
1800 4 }
1801 4 }
1802 4 }
1803 4 }
1804 4 }
1805 4 }
1806 4 }
1807 4 }
1808 4 }
1809 4 }
1810 4 }
1811 4 }
1812 4 }
1813 4 }
1814 4 }
1815 4 }
1816 4 }
1817 4 }
1818 4 }
1819 4 }
1820 4 }
1821 4 }
1822 4 }
1823 4 }
1824 4 }
1825 4 }
1826 4 }
1827 4 }
1828 4 }
1829 4 }
1830 4 }
1831 4 }
1832 4 }
1833 4 }
1834 4 }
1835 4 }
1836 4 }
1837 4 }
1838 4 }
1839 4 }
1840 4 }
1841 4 }
1842 4 }
1843 4 }
1844 4 }
1845 4 }
1846 4 }
1847 4 }
1848 4 }
1849 4 }
1850 4 }
1851 4 }
1852 4 }
1853 4 }
1854 4 }
1855 4 }
1856 4 }
1857 4 }
1858 4 }
1859 4 }
1860 4 }
1861 4 }
1862 4 }
1863 4 }
1864 4 }
1865 4 }
1866 4 }
1867 4 }
1868 4 }
1869 4 }
1870 4 }
1871 4 }
1872 4 }
1873 4 }
1874 4 }
1875 4 }
1876 4 }
1877 4 }
1878 4 }
1879 4 }
1880 4 }
1881 4 }
1882 4 }
1883 4 }
1884 4 }
1885 4 }
1886 4 }
1887 4 }
1888 4 }
1889 4 }
1890 4 }
1891 4 }
1892 4 }
1893 4 }
1894 4 }
1895 4 }
1896 4 }
1897 4 }
1898 4 }
1899 4 }
1900 4 }

```



```

1771 3         filter_cmd.fdcout = 1;
1772 2         (void)debugf(cmd_fd(0), 1);
1773 2     }
1774 2     else
1775 3     {
1776 3         filter_cmd.fdcout = rcmd_fd(0);
1777 3         (void) sprintf(fdcap, filter_cmd.argv[filter_cmd.fdcinfo],
1778 3             filter_cmd.fdcout);
1779 3         filter_cmd.argv[filter_cmd.fdcinfo] = fdcap;
1780 2     }
1781 2
1782 2     /* Send read uid string on filter_cmd.fdcout to client process
1783 2     */
1784 2     z = (int)section(human_name);
1785 2     if ((n == looprev)
1786 2         || filter_cmd.fdcout, human_name, 1, write_CMD_no_sintr) != 1)
1787 3     {
1788 3         rthe_log_stats(BREKCOVER, JKERR(errno));
1789 3         /* Can't write fd bytes to host '\$ay', "
1790 3             "cmd, errno=fd, fd=fdyn", 1, rcmd_stuff(0,
1791 3                 filter_cmd.fdcout);
1792 3         }
1793 3         -exit(2);
1794 3     }
1795 3     if ((n == looprev)
1796 3         || filter_cmd.fdcout, "\n", 1, write_CMD_no_sintr) != 1)
1797 3     {
1798 3         rthe_log_stats(BREKCOVER, JKERR(errno));
1799 3         /* Can't write fd bytes to host '\$ay', "
1800 3             "cmd, errno=fd, fd=fdyn", 1, rcmd_stuff(0,
1801 3                 filter_cmd.fdcout);
1802 3         }
1803 3         -exit(3);
1804 3     }
1805 2     (void)close(xcplogopen_pipe(0));
1806 2     (void)dup2(xcplogopen_pipe(1), 2);
1807 2     (void)close(xcplogopen_pipe(1));
1808 2
1809 2     /* Call the CUD layer so we can warn SQL that an execvp
1810 2     is coming. If this isn't an SQL socket this call is a
1811 2     no-op
1812 2     */
1813 2     (void)CUD_execvpexecvp(filter_cmd.fdcout);
1814 2
1815 2     (void)execvp(filter_cmd, filter_cmd.argv); /* run xcplogopen */
1816 2     rthe_log_stats(BREKCOVER, JKERR(errno)); /* Can't exec '\$ay', errno=fdyn",
1817 2         filter_cmd(errno);
1818 2     break;
1819 2
1820 2     default:
1821 2     /* parent */
1822 2     (void)close(xcplogopen_pipe(1));
1823 2     (void)close(xcplogopen_pipe(0));
1824 2     /* end of switch */
1825 2 }
1826 1
1827 1 /* Parent has no use for this file descriptor any more,
1828 1 and nuking it
1829 1
1830 1
1831 1
1832 1
1833 1
1834 1
1835 1
1836 1
1837 1
1838 1
1839 1
1840 1
1841 1
1842 1
1843 1
1844 1
1845 1
1846 1
1847 1
1848 1
1849 1
1850 1
1851 1
1852 1
1853 1
1854 1
1855 2
1856 2
1857 3
1858 3
1859 4
1860 4
1861 4
1862 4
1863 4
1864 4
1865 4
1866 4
1867 5
1868 5
1869 5
1870 4
1871 4
1872 3
1873 2
1874 1
1875 1
1876 1
1877 2
1878 2
1879 2
1880 2
1881 2
1882 2
1883 2
1884 2
1885 2
1886 2
1887 2
1888 3
1889 3
1890 3

```

```

1891 1     /* now is important so that if the child dies while the rcmd still
1892 1     * wants more input from the child, the rcmd will die too !
1893 1     * hanging around hoping that maybe this parent process will supply
1894 1     * the data) */
1895 1
1896 1     /* we didn't really dup these if they were SFP sockets so
1897 1     * avoid the closes in that case
1898 1     if ((pval->val.&programname)
1899 1         {
1900 1         if (rcmd_fd(0) != -1)
1901 1         {
1902 1             (void)close(rcmd_fd(0));
1903 1         }
1904 1         if (rcmd_fd(1) != -1)
1905 1         {
1906 1             (void)close(rcmd_fd(1));
1907 1         }
1908 1     }
1909 1
1910 1     /* However in the case of parent we did get a network
1911 1     * socket, we should need to issue one
1912 1     * close on the one SFP socket. We can detect that
1913 1     case because rcmd_stderr will not be equal to rcmd_fd(0)
1914 1     */
1915 1     if (rcmd_stderr != rcmd_fd(0))
1916 1     {
1917 1         if (rcmd_fd(0) != -1)
1918 1         {
1919 1             /* If not a SFP socket, do the close, if not
1920 1             * we close SFP socket by calling sprintf()
1921 1             * data has been moved by calling sprintf()
1922 1             */
1923 1             if (0 == CUD_setSocket(rcmd_fd(0)) )
1924 1             {
1925 1                 (void) CUD_close(rcmd_fd(0));
1926 1             }
1927 1             rcmd_fd(0) = -1;
1928 1         }
1929 1     }
1930 1
1931 1     if (0 == strcmp(method, "set") || (0 == strcmp(
1932 1         method, "editlink")) )
1933 1     {
1934 1         /* Because of occasional loss of data due to cloning issues when
1935 1         * connected via editlink, turn on keepalive. In version 4.7 this will
1936 1         * also be addressed by editlink, directly. (OSGWS38596)
1937 1         */
1938 1         if (-1 == setsockopt(rcmd_sock,
1939 1             SOL_SOCKET,
1940 1             TCP_KEEPAIVE,
1941 1             (char *)"1600",
1942 1             sizeof(int)) )
1943 1         {
1944 1             rthe_log_stats(BREKCOVER, JKERR(errno),
1945 1                 "warning: setsockopt for SO_KEEPAIVE
1946 1                 failed");
1947 1         }
1948 1     }
1949 1
1950 1
1951 1
1952 1
1953 1
1954 1
1955 1
1956 1
1957 1
1958 1
1959 1
1960 1
1961 1
1962 1
1963 1
1964 1
1965 1
1966 1
1967 1
1968 1
1969 1
1970 1
1971 1
1972 1
1973 1
1974 1
1975 1
1976 1
1977 1
1978 1
1979 1
1980 1
1981 1
1982 1
1983 1
1984 1
1985 1
1986 1
1987 1
1988 1
1989 1
1990 1

```

[illegible]

Mon Oct 13 16:00:57 2008	z_confiderr	Page 102 of 134
1955 2	rcmd_saddr;	
1956 2	rcmd_serv[0];	
1957 2	xcprocgen_prog_fd,	
1958 2	xcprocgen_pid,	
1959 2	remote_exitinfo;	
1960 2		
1961 2	if(! rcmd_serv)	
1962 2	0, "error monitoring Auzproc's children.")*/	
1963 2	}	
1964 2	/*	
1965 2	notify the main program of remote success/failure	
1966 2	*/	
1967 2		
1968 2	c = 'R';	
1969 2	s = sizeof(remote_exitinfo);	
1970 2	write_or_die(cxp->sg_w_fd, &c, 1, _exit);	
1971 2	write_or_die(cxp->sg_w_fd, (char *)&c, sizeof i, _exit);	
1972 2	write_or_die(cxp->sg_w_fd,	
1973 2	char *)&remote_exitinfo, 1, _exit);	
1974 2		
1975 2	if(! 0	
1976 2	rcmd_status(0, "Auzproc(%d) remote exit is %d", getpid(),	
1977 2	remote_exitinfo);	
1978 2		
1979 2	/*	
1980 2	is our connection method anything BUT "newware"?	
1981 2	*/	
1982 2	if (0 != strcmp(method, "newware"))	
1983 2	/*	
1984 2	Now wait for the exit code of the filter (local process	
1985 2	*/	
1986 2	waitpid(1, my return with error before the child	
1987 2	exists in that case, we want to continue	
1988 2	with the wait. (fix for OSSM15487)	
1989 2	*/	
1990 2		
1991 2	while (waitpid(xcprocgen_pid, &wait_result, 0) == -1) {&	
1992 2	{ EINTR == error() }	
1993 2	{	
1994 2	/* empty while loop */	
1995 2	}	
1996 2	/*	
1997 2	extended while loop either because the return value of	
1998 2	waitpid is NOT -1, or the error is NOT EINTR	
1999 2	*/	
2000 2		
2001 2	else	
2002 2	{	
2003 2	/*	
2004 2	try's newware */	
2005 2	/*	
2006 2	***** HACK *** HACK *** HACK *** HACK	
2007 2	***** HACK *** HACK *** HACK *** HACK	
2008 2	***** HACK *** HACK *** HACK *** HACK	
2009 2	***** HACK *** HACK *** HACK *** HACK	
2010 2	***** HACK *** HACK *** HACK *** HACK	
2011 2	***** HACK *** HACK *** HACK *** HACK	
2012 2	***** HACK *** HACK *** HACK *** HACK	
2013 2	***** HACK *** HACK *** HACK *** HACK	
2014 2	***** HACK *** HACK *** HACK *** HACK	
2015 2	***** HACK *** HACK *** HACK *** HACK	
2016 2	***** HACK *** HACK *** HACK *** HACK	
2017 2	***** HACK *** HACK *** HACK *** HACK	
2018 2	***** HACK *** HACK *** HACK *** HACK	
2019 2	***** HACK *** HACK *** HACK *** HACK	
2020 2	***** HACK *** HACK *** HACK *** HACK	
2021 2	***** HACK *** HACK *** HACK *** HACK	
2022 2	***** HACK *** HACK *** HACK *** HACK	
2023 2	***** HACK *** HACK *** HACK *** HACK	
2024 2	***** HACK *** HACK *** HACK *** HACK	
2025 2	***** HACK *** HACK *** HACK *** HACK	
2026 2	***** HACK *** HACK *** HACK *** HACK	
2027 2	***** HACK *** HACK *** HACK *** HACK	
2028 2	***** HACK *** HACK *** HACK *** HACK	
2029 2	***** HACK *** HACK *** HACK *** HACK	
2030 2	***** HACK *** HACK *** HACK *** HACK	
2031 2	***** HACK *** HACK *** HACK *** HACK	
2032 2	***** HACK *** HACK *** HACK *** HACK	
2033 2	***** HACK *** HACK *** HACK *** HACK	
2034 2	***** HACK *** HACK *** HACK *** HACK	
2035 2	***** HACK *** HACK *** HACK *** HACK	
2036 2	***** HACK *** HACK *** HACK *** HACK	
2037 2	***** HACK *** HACK *** HACK *** HACK	
2038 2	***** HACK *** HACK *** HACK *** HACK	
2039 2	***** HACK *** HACK *** HACK *** HACK	
2040 2	***** HACK *** HACK *** HACK *** HACK	
2041 2	***** HACK *** HACK *** HACK *** HACK	
2042 2	***** HACK *** HACK *** HACK *** HACK	
2043 2	***** HACK *** HACK *** HACK *** HACK	
2044 2	***** HACK *** HACK *** HACK *** HACK	
2045 2	***** HACK *** HACK *** HACK *** HACK	
2046 2	***** HACK *** HACK *** HACK *** HACK	
2047 2	***** HACK *** HACK *** HACK *** HACK	
2048 2	***** HACK *** HACK *** HACK *** HACK	
2049 2	***** HACK *** HACK *** HACK *** HACK	
2050 2	***** HACK *** HACK *** HACK *** HACK	
2051 2	***** HACK *** HACK *** HACK *** HACK	
2052 2	***** HACK *** HACK *** HACK *** HACK	
2053 2	***** HACK *** HACK *** HACK *** HACK	
2054 2	***** HACK *** HACK *** HACK *** HACK	
2055 2	***** HACK *** HACK *** HACK *** HACK	
2056 2	***** HACK *** HACK *** HACK *** HACK	
2057 2	***** HACK *** HACK *** HACK *** HACK	
2058 2	***** HACK *** HACK *** HACK *** HACK	
2059 2	***** HACK *** HACK *** HACK *** HACK	
2060 2	***** HACK *** HACK *** HACK *** HACK	
2061 2	***** HACK *** HACK *** HACK *** HACK	
2062 2	***** HACK *** HACK *** HACK *** HACK	
2063 2	***** HACK *** HACK *** HACK *** HACK	</


```

2011 3 * This hack is basically for Netware.
2012 3 * does not send anything back to the ( Network TCP/IP close()
2013 3 * that is running as pid pfc0_pid. So the "filler" process
2014 3 * just keeps on sending data across the network and
2015 3 * anybody there to read it.
2016 3 * Therefore, "auxproc" doesn't finish until all data has
2017 3 * been written to the socket even though no process is
2018 3 * reading
2019 3 * it. However,
2020 3 * the process might hang indefinitely if the network
2021 3 * buffer(s) become(s) full.
2022 3 * This is how we will get around this.
2023 3 * not think of a better way. None of the normal TCP/IP
2024 3 * mechanisms seem to work with Netware.
2025 3 * If the exitInfo !
2026 3 * failure() reports failure...
2027 3 * After we have received the exit info from the Netware
2028 3 * then try to get it's "wait_result" twice with a 2
2029 3 * in between. If we don't get it,
2030 3 * process is hung and can't wind down.
2031 3 * Therefore,
2032 3 * STOPPED, let's send a STOPPED to the process. I chose
2033 3 * STOPPED because this is usually trapped by our
2034 3 * handled in a controlled manner.
2035 3 * it, attempting for 2 seconds. Just keep on trying to kill
2036 3 * it DOES go away.
2037 3 * Just keep on trying to kill it,
2038 3 * then see if it went away until it DOES go away.
2039 3 * Apparently, the kill()
2040 3 * where is actively counting like positioning the tape.
2041 3 * the reason for having to do the kill()
2042 3 * works.
2043 3 *
2044 3 * if (remote_exitInfo)
2045 3 * /**
2046 3 * ** We received an error exit status from Novell
2047 3 * **
2048 3 * int exitInfo;

```

```

2057 4 exitInfo = waitpid(
2058 4 while (exitInfo)
2059 4 {
2060 4 * Wait a couple of seconds longer and try again
2061 4 *
2062 4 sleep(2);
2063 4 exitInfo = waitpid(
2064 4 xprocogen_pid, &wait_result, WNOHANG);
2065 4
2066 4 if (exitInfo)
2067 4 {
2068 4 kill(xprocogen_pid, SIGTERM); /* BLAST IT! */
2069 4 }
2070 4 }
2071 4 else
2072 4 {
2073 4 * We received a success return status from Novell
2074 4 **
2075 4 **
2076 4 * waitpid() may return with EINTR before the child
2077 4 * proc has finished. So we will continue
2078 4 * to call wait. (fix for 0506w15487)
2079 4 while (waitpid(
2080 4 xprocogen_pid, &wait_result, 0) == -1) &&
2081 4 (EINTR == errno)
2082 4 {
2083 4 * empty while loop */
2084 4 }
2085 4 * exited while loop either because the return value
2086 4 * waitpid is NOT -1, or the errno is NOT EINTR
2087 4 )
2088 4 {
2089 4 * main command loop in caller will send this result for us
2090 4 *
2091 4 *
2092 4 * wait_result contains the exit status of the child
2093 4 * process. Use the standard macros to determine the
2094 4 * status of the child process and see the return value
2095 4 * to indicate the status accurately.
2096 4 *
2097 4 * if (WIFEXITED(
2098 4 wait_result)) /* child proc exited normally */
2099 4 {
2100 4 * get the result to the exit code
2101 4 *
2102 4 *
2103 4 * wait_result = WEXITSTATUS(wait_result);
2104 4 }
2105 4 }

```

```

Mon Oct 13 16:00:37 2008                                Z. rcmidiller                                Page 106 of 134

2116 2
2117 3
2118 4
2119 5
2120 6
2121 7
2122 8
2123 9
2124 10
2125 11
2126 12
2127 13
2128 14
2129 15
2130 16
2131 17
2132 18
2133 19
2134 20
2135 21
2136 22
2137 23
2138 24
2139 25
2140 26
2141 27
2142 28
2143 29
2144 30
2145 31
2146 32
2147 33
2148 34
2149 35
2150 36
2151 37
2152 38
2153 39
2154 40
2155 41
2156 42
2157 43
2158 44
2159 45
2160 46
2161 47
2162 48
2163 49
2164 50
2165 51
2166 52
2167 53
2168 54
2169 55
2170 56
2171 57
2172 58
2173 59
2174 60
2175 61
2176 62
2177 63
2178 64
2179 65
2180 66
2181 67
2182 68
2183 69
2184 70
2185 71
2186 72
2187 73
2188 74
2189 75
2190 76
2191 77
2192 78
2193 79
2194 80
2195 81
2196 82
2197 83
2198 84
2199 85
2200 86
2201 87
2202 88
2203 89
2204 90
2205 91
2206 92
2207 93
2208 94
2209 95
2210 96
2211 97
2212 98
2213 99
2214 100
2215 101
2216 102
2217 103
2218 104
2219 105
2220 106
2221 107
2222 108
2223 109
2224 110
2225 111
2226 112
2227 113
2228 114
2229 115
2230 116
2231 117
2232 118
2233 119
2234 120
2235 121
2236 122
2237 123
2238 124
2239 125
2240 126
2241 127
2242 128
2243 129
2244 130
2245 131
2246 132
2247 133
2248 134
2249 135
2250 136
2251 137
2252 138
2253 139
2254 140
2255 141
2256 142
2257 143
2258 144
2259 145
2260 146
2261 147
2262 148
2263 149
2264 150
2265 151
2266 152
2267 153
2268 154
2269 155
2270 156
2271 157
2272 158
2273 159
2274 160
2275 161
2276 162
2277 163
2278 164
2279 165
2280 166
2281 167
2282 168
2283 169
2284 170
2285 171
2286 172
2287 173
2288 174
2289 175
2290 176
2291 177
2292 178
2293 179
2294 180
2295 181
2296 182
2297 183
2298 184
2299 185
2300 186
2301 187
2302 188
2303 189
2304 190
2305 191
2306 192
2307 193
2308 194
2309 195
2310 196
2311 197
2312 198
2313 199
2314 200
2315 201
2316 202
2317 203
2318 204
2319 205
2320 206
2321 207
2322 208
2323 209
2324 210
2325 211
2326 212
2327 213
2328 214
2329 215
2330 216
2331 217
2332 218
2333 219
2334 220
2335 221
2336 222
2337 223
2338 224
2339 225
2340 226
2341 227
2342 228
2343 229
2344 230
2345 231
2346 232
2347 233
2348 234
2349 235
2350 236
2351 237
2352 238
2353 239
2354 240
2355 241
2356 242
2357 243
2358 244
2359 245
2360 246
2361 247
2362 248
2363 249
2364 250
2365 251
2366 252
2367 253
2368 254
2369 255
2370 256
2371 257
2372 258
2373 259
2374 260
2375 261
2376 262
2377 263
2378 264
2379 265
2380 266
2381 267
2382 268
2383 269
2384 270
2385 271
2386 272
2387 273
2388 274
2389 275
2390 276
2391 277
2392 278
2393 279
2394 280
2395 281
2396 282
2397 283
2398 284
2399 285
2400 286
2401 287
2402 288
2403 289
2404 290
2405 291
2406 292
2407 293
2408 294
2409 295
2410 296
2411 297
2412 298
2413 299
2414 300
2415 301
2416 302
2417 303
2418 304
2419 305
2420 306
2421 307
2422 308
2423 309
2424 310
2425 311
2426 312
2427 313
2428 314
2429 315
2430 316
2431 317
2432 318
2433 319
2434 320
2435 321
2436 322
2437 323
2438 324
2439 325
2440 326
2441 327
2442 328
2443 329
2444 330
2445 331
2446 332
2447 333
2448 334
2449 335
2450 336
2451 337
2452 338
2453 339
2454 340
2455 341
2456 342
2457 343
2458 344
2459 345
2460 346
2461 347
2462 348
2463 349
2464 350
2465 351
2466 352
2467 353
2468 354
2469 355
2470 356
2471 357
2472 358
2473 359
2474 360
2475 361
2476 362
2477 363
2478 364
2479 365
2480 366
2481 367
2482 368
2483 369
2484 370
2485 371
2486 372
2487 373
2488 374
2489 375
2490 376
2491 377
2492 378
2493 379
2494 380
2495 381
2496 382
2497 383
2498 384
2499 385
2500 386
2501 387
2502 388
2503 389
2504 390
2505 391
2506 392
2507 393
2508 394
2509 395
2510 396
2511 397
2512 398
2513 399
2514 400
2515 401
2516 402
2517 403
2518 404
2519 405
2520 406
2521 407
2522 408
2523 409
2524 410
2525 411
2526 412
2527 413
2528 414
2529 415
2530 416
2531 417
2532 418
2533 419
2534 420
2535 421
2536 422
2537 423
2538 424
2539 425
2540 426
2541 427
2542 428
2543 429
2544 430
2545 431
2546 432
2547 433
2548 434
2549 435
2550 436
2551 437
2552 438
2553 439
2554 440
2555 441
2556 442
2557 443
2558 444
2559 445
2560 446
2561 447
2562 448
2563 449
2564 450
2565 451
2566 452
2567 453
2568 454
2569 455
2570 456
2571 457
2572 458
2573 459
2574 460
257
```

Mon Oct 13 16:00:37 2008	decodecookie	Page 106 of 134
2179	static enum input_states decodecookie(char *cp) {	
2181 1	struct cookiestate {	
2182 1	char *cookie;	
2183 1	enum input_states state;	
2184	};	
2186 1	static struct cookiestate c2sb1[] = {	INSTATE_COPY_TO STDOUT ,
2187 1	{ REMFD_COOKIE_F2OUT2,	INSTATE_COPY_NO STDOUT ,
2188 1	{ REMFD_COOKIE_F2OUT2,	INSTATE_COPY_NO STDOUT ,
2189 1	{ REMFD_COOKIE_F2OUTEND,	INSTATE_COPY_NO STDOUT ,
2190 1	{ REMFD_COOKIE_STATUS,	INSTATE_COPY_NO STDOUT ,
2191 1	{ NULL,	INSTATE_COPY_NO STDOUT ,
2192 1	};	
2195	static enum input_states	
2196	decodecookie(char *cp)	
2197 1	{	
2198 1	struct cookiestate *c2s;	
2201 1	for (c2s = c2sb1; c2s->cookie != NULL; c2s++)	
2202 2	{	
2203 2	if (strcmp(c2s->cookie, cp) == 0)	
2204 3	return c2s->state;	
2205 3	}	
2206 1	return INSTATE_SEARCH_PREFIX0;	
2207 1	}	
2208 1	/* end of decodecookie */	
2209	}	

Mon Oct 13 16:00:37 2008

RSI.laurnatic.40

Page 106 of 134

```

2211 static void
2212 sigusr1_handler(int sigvalue)
2213 {
2214     if (debugmode)
2215         the_log_stats(0, "USR1 (ATTN) signal received!\n");
2216 }
2217
2218 if (0 < xcplogon_pid)
2219     { void)kill(xcplogon_pid, SIGTERM);
2220 }
2221
2222 /*+attr,oci
2223 /* end of sigusr1_handler() */

```

```

2227 static void
2228 sigterm_handler(int sigvalue)
2229 {
2230     if (debugmode)
2231         the_log_stats(0, "TERM signal received!\n");
2232 }
2233
2234 the_close_logs(logging_channel);
2235
2236 /* ESS workaround: Make a call to CDL_exit
2237 if symmpath workitem and CDL_exit has
2238 not already been called. This is to clean
2239 up sockets since we have made sure we
2240 CDL_socketexit, we have to make sure we
2241 clean up */
2242 if ( !is_symmpath) && (0==sp_colexitedone) )
2243     {
2244         CDL_exit();
2245         sp_colexitedone = 1;
2246     }
2247
2248 signal(SIGTERM, SIG_DFL); /* restore default (terminate) action */
2249 (void)kill(getpid(), SIGTERM);
2250
2251 /* end of sigterm_handler() */
2252
2253

```

```

2256 /*
2257  * Invoked when running recover in debugging mode.
2258  * This execs a separate a.out file to implement the auxproc, so that
2259  * breakpoints can be set in the recover code without affecting
2260  * the auxproc code.
2261  */
2262
2263 static void
2264 z_exec_separate_auxproc(struct auxproc_context *cxp,
2265                          char *pathname)
2266 {
2267     char r_fd_set[32];
2268     char w_fd_set[32];
2269     char r_bulk_fd_set[32];
2270     char r_bulk_fd_set[32];
2271     char dbgmodestr[32];
2272     char argv0;
2273
2274     /*
2275      * NOTE: This name is "special", the recover main()
2276      * function looks for it to know when it's being invoked
2277      * to perform auxproc processing.
2278      */
2279     argv0 = "zbr_auxproc";
2280
2281     (void) sprintf(procnum_str, "%d", cxp->ap.my_auxnum);
2282     (void) sprintf(w_fd_set, "%d", cxp->w_fd);
2283     (void) sprintf(r_fd_set, "%d", cxp->r_fd);
2284     (void) sprintf(r_bulk_fd_set, "%d", cxp->r_bulk_fd);
2285     (void) sprintf(dbgmodestr, "%d", cxp->dbgmod);
2286     (void) strcpy(pathname, argv0);
2287
2288     /* prog to execute */
2289     argv0 = argv 0;
2290     procnum_str = argv 1;
2291     w_fd_set = argv 2;
2292     r_fd_set = argv 3;
2293     r_bulk_fd_set = argv 4;
2294     dbgmodestr = argv 5;
2295     (char *)0;
2296
2297     /* If we get here, the exec did not go. Caller will bomb for us.
2298      */
2299     /* end of z_exec_separate_auxproc() */
2300 }

```

```

2303 /*
2304  * Use these functions with loops to build
2305  * a loop read (or loop write) that ignores EINTR.
2306  */
2307
2308 static int
2309 read_CDL_no_eintr(int fd,
2310                  char *buf,
2311                  int nbytes)
2312 {
2313     int r;
2314
2315     do
2316     {
2317         errno = 0;
2318         r = CDL_Read(fd, buf, (uint_t)nbytes, 0);
2319         while (r == -1 && errno == EINTR);
2320     }
2321     return r;
2322 }

```

```

2326 static int
2327 write_CD_L_no_sincr(int fd,
2328                     char *buf,
2329                     int nbytes)
2330 {
2331     int r;
2332     do
2333     {
2334         errno = 0;
2335         r = CD_L_write(fd, buf, (uint_t)nbytes, 0);
2336     } while (r == -1 && errno == EINTR);
2337     return r;
2338 } /* end of write_CD_L_no_sincr() */

```

```

3342 /* Malloc, interruptibly
3343    * for at least one byte to become available on fd.
3344    * Returns 0 if at least one byte is available.
3345    * Returns -1 for any type of failure, including wait interruption.
3346    * Sets errno appropriately when -1 is returned.
3347    */
3348 int
3349 fd_avail_1_wait_int(int fd)
3350 {
3351     esl_fdsel_t rcbits;
3352     E_FD_ZERO(&rcbits);
3353     E_FD_SET(fd, &rcbits);
3354     /* Don't need to examine rcbits after select, since only one
3355        * fd is selected, therefore return value can be computed
3356        * directly from select return value.
3357        */
3358     if (esl_select(E_FD_SETSIZE, &rcbits, NULL, NULL, NULL) == -1)
3359     {
3360         return -1;
3361     }
3362     return 0;
3363 } /* end of fd_avail_1_wait_int() */

```

```

2372  /*
2373  * Test, interruptibly,
2374  *   for at least one byte to become available on fd.
2375  * Returns 1 if at least one byte is available.
2376  * Returns -1 for any type of failure, including test interruption.
2377  * Return 0 if no data is available.
2378  */
2379  * Sets errno appropriate when -1 is returned.
2380  */
2381
2382  int
2383  fd_await_test_interrupt(int fd)
2384  {
2385
2386      int retStatus;
2387      struct pollfd fd_test;
2388
2389      fd_test.fd = fd;
2390      fd_test.events = POLLIN;
2391      fd_test.revents = 0;
2392
2393      /*
2394       * Don't need to examine rbits after select, since only one
2395       * fd is in this set -- therefore return value can be computed
2396       * directly from select return value.
2397       */
2398
2399      if ((retStatus = CDB_poll_read(&fd_test, 1, 0)) == -1)
2400      {
2401          /* ERROR encountered */
2402          return -1;
2403      }
2404      return retStatus;
2405  }
2406
2407  /* end of fd_await_test_int.c */

```

```

2409  /*
2410  * Test, for at least one byte to become available on fd.
2411  * Returns 1 if at least one byte is available.
2412  * Returns -1 for any type of failure other than EINTR.
2413  * Return 0 if no data is available.
2414  */
2415  * Sets errno appropriate when -1 is returned.
2416  */
2417
2418  int fd_await_test1(int fd)
2419  {
2420      int retStatus;
2421      struct pollfd fd_test;
2422
2423      fd_test.fd = fd;
2424      fd_test.events = POLLIN;
2425      fd_test.revents = 0;
2426
2427      /*
2428       * Don't need to examine rbits after select, since only one
2429       * fd is in this set -- therefore return value can be computed
2430       * directly from select return value.
2431       */
2432
2433      while ((-1 == (retStatus = CDB_poll_read(&fd_test, 1, 0))) &&
2434             (EINTR == errno))
2435      {
2436          ;
2437      }
2438      return retStatus;
2439  }
2440
2441

```

[illegible]

Mon Oct 13 16:00:37 2008	git_difft_tool	Page 116 of 134
2550 1	{ if (pid == ERR_PORK(1) == 0)	
2550 2	{	
2550 3	/*	child processing goes here while parent is stopped! If vfork() fails, setup stdio to work using the pipes
2560 2	/*	
2560 3	if ((close(0)<0) (close(1)<0) (close(2)<0))	
2560 4	writeProcStrings(cop->wp, w_proc_fd,	
2560 5	EMBERPROG_AUXPROC_ERROR, 0,	
2560 6	unable to close std	
2560 7	in/out/err) for forked childn*);	
2560 8	_exit(-1);	
2560 9	if ((dup(0)<0) (dup(1)<0) (dup(2)<0))	
2560 10	writeProcStrings(cop->wp, w_proc_fd,	
2560 11	EMBERPROG_AUXPROC_ERROR, 0,	
2560 12	unable to dup pipe ends for forked childn*);	
2560 13	_exit(-1);	
2560 14	if ((close(x[0])<0) (close(y[0])<0) (close(z[0])<0)	
2560 15	(close(x[1])<0) (close(y[1])<0) (close(z[1])<0))	
2560 16	{	
2560 17	writeProcStrings!	
2560 18	cop->wp, w_proc_fd, EMBERPROG_AUXPROC_ERROR, 0,	
2560 19	unable to close pipe ends for forked childn*);	
2560 20	_exit(-1);	
2560 21	/*	At this moment the process has a real UID of the user executing
2560 22	* X librecovery and an effective UID of root (X librecovery has	
2560 23	* the stuid bit set and is owned by root).	
2560 24	* The first problem is the access/	
2560 25	* function call, uses the real UID	
2560 26	* of the process. So if the user is not root or daemon,	
2560 27	* function will fail. This causes the recovery to fail.	
2560 28	* problem 1.	
2560 29	* The second problem is System V Unix does not pass the	
2560 30	* to processes doing an exec of a shell.	
2560 31	* Instead the effective UID	
2560 32	* is changed back to the real UID of the process. However,	
2560 33	* implementation does maintain the effective UID of the	
2560 34	* process.	
2560 35	* On System V,	
2560 36	* neither the real or effective UID will be root during	
2560 37	* the exec of the shell script if the user executing	
2560 38	* X librecovery is	
2560 39	* a non-root user. This process will not be able to perform	
2560 40	* /bin/rm -c	
2560 41	/*/systemd/bin/startup because this processes real	
2560 42	FSID=umain@1c50	

Mon Oct 13 16:00:37 2008

Page 116 of 134


```

2666 1         if (istat == 0)
2667 2         {
2668 2             sigaction(SIGCHLD, &old_act, NULL);
2669 1         }
2670 1         return ebr_direct_rand_fids;
2671 1         /* end of ebr_direct_rand() */
2672 1     }

```

```

/*
 * routine to search the clients installed file for a clients
 * method. Returns a pointer to the connection method string if
 * successful,
 * NULL if the client entry could not be found.
 */

```

```

2680 static char *
2681 rb_getmethod(register char *host,
2682              int c *concurrency,
2683              int c *client_type)
2684 {
2685     FILE
2686     *fp;
2687     FILE
2688     *fp2;
2689     FILE
2690     *fp3;
2691     FILE
2692     *fp4;
2693     FILE
2694     *fp5;
2695     FILE
2696     *fp6;
2697     FILE
2698     *fp7;
2699     FILE
2700     *fp8;
2701     FILE
2702     *fp9;
2703     FILE
2704     *fp10;
2705     FILE
2706     *fp11;
2707     FILE
2708     *fp12;
2709     FILE
2710     *fp13;
2711     FILE
2712     *fp14;
2713     FILE
2714     *fp15;
2715     FILE
2716     *fp16;
2717     FILE
2718     *fp17;
2719     FILE
2720     *fp18;
2721     FILE
2722     *fp19;
2723     FILE
2724     *fp20;
2725     FILE
2726     *fp21;
2727     FILE
2728     *fp22;
2729     FILE
2730     *fp23;
2731     FILE
2732     *fp24;
2733     FILE
2734     *fp25;
2735     FILE
2736     *fp26;
2737     FILE
2738     *fp27;
2739     FILE
2740     *fp28;
2741     FILE
2742     *fp29;
2743     FILE
2744     *fp30;
2745     FILE
2746     *fp31;
2747     FILE
2748     *fp32;
2749     FILE
2750     *fp33;
2751     FILE
2752     *fp34;
2753     FILE
2754     *fp35;
2755     FILE
2756     *fp36;
2757     FILE
2758     *fp37;
2759     FILE
2760     *fp38;
2761     FILE
2762     *fp39;
2763     FILE
2764     *fp40;
2765     FILE
2766     *fp41;
2767     FILE
2768     *fp42;
2769     FILE
2770     *fp43;
2771     FILE
2772     *fp44;
2773     FILE
2774     *fp45;
2775     FILE
2776     *fp46;
2777     FILE
2778     *fp47;
2779     FILE
2780     *fp48;
2781     FILE
2782     *fp49;
2783     FILE
2784     *fp50;
2785     FILE
2786     *fp51;
2787     FILE
2788     *fp52;
2789     FILE
2790     *fp53;
2791     FILE
2792     *fp54;
2793     FILE
2794     *fp55;
2795     FILE
2796     *fp56;
2797     FILE
2798     *fp57;
2799     FILE
2800     *fp58;
2801     FILE
2802     *fp59;
2803     FILE
2804     *fp60;
2805     FILE
2806     *fp61;
2807     FILE
2808     *fp62;
2809     FILE
2810     *fp63;
2811     FILE
2812     *fp64;
2813     FILE
2814     *fp65;
2815     FILE
2816     *fp66;
2817     FILE
2818     *fp67;
2819     FILE
2820     *fp68;
2821     FILE
2822     *fp69;
2823     FILE
2824     *fp70;
2825     FILE
2826     *fp71;
2827     FILE
2828     *fp72;
2829     FILE
2830     *fp73;
2831     FILE
2832     *fp74;
2833     FILE
2834     *fp75;
2835     FILE
2836     *fp76;
2837     FILE
2838     *fp77;
2839     FILE
2840     *fp78;
2841     FILE
2842     *fp79;
2843     FILE
2844     *fp80;
2845     FILE
2846     *fp81;
2847     FILE
2848     *fp82;
2849     FILE
2850     *fp83;
2851     FILE
2852     *fp84;
2853     FILE
2854     *fp85;
2855     FILE
2856     *fp86;
2857     FILE
2858     *fp87;
2859     FILE
2860     *fp88;
2861     FILE
2862     *fp89;
2863     FILE
2864     *fp90;
2865     FILE
2866     *fp91;
2867     FILE
2868     *fp92;
2869     FILE
2870     *fp93;
2871     FILE
2872     *fp94;
2873     FILE
2874     *fp95;
2875     FILE
2876     *fp96;
2877     FILE
2878     *fp97;
2879     FILE
2880     *fp98;
2881     FILE
2882     *fp99;
2883     FILE
2884     *fp100;
2885     FILE
2886     *fp101;
2887     FILE
2888     *fp102;
2889     FILE
2890     *fp103;
2891     FILE
2892     *fp104;
2893     FILE
2894     *fp105;
2895     FILE
2896     *fp106;
2897     FILE
2898     *fp107;
2899     FILE
2900     *fp108;
2901     FILE
2902     *fp109;
2903     FILE
2904     *fp110;
2905     FILE
2906     *fp111;
2907     FILE
2908     *fp112;
2909     FILE
2910     *fp113;
2911     FILE
2912     *fp114;
2913     FILE
2914     *fp115;
2915     FILE
2916     *fp116;
2917     FILE
2918     *fp117;
2919     FILE
2920     *fp118;
2921     FILE
2922     *fp119;
2923     FILE
2924     *fp120;
2925     FILE
2926     *fp121;
2927     FILE
2928     *fp122;
2929     FILE
2930     *fp123;
2931     FILE
2932     *fp124;
2933     FILE
2934     *fp125;
2935     FILE
2936     *fp126;
2937     FILE
2938     *fp127;
2939     FILE
2940     *fp128;
2941     FILE
2942     *fp129;
2943     FILE
2944     *fp130;
2945     FILE
2946     *fp131;
2947     FILE
2948     *fp132;
2949     FILE
2950     *fp133;
2951     FILE
2952     *fp134;
2953     FILE
2954     *fp135;
2955     FILE
2956     *fp136;
2957     FILE
2958     *fp137;
2959     FILE
2960     *fp138;
2961     FILE
2962     *fp139;
2963     FILE
2964     *fp140;
2965     FILE
2966     *fp141;
2967     FILE
2968     *fp142;
2969     FILE
2970     *fp143;
2971     FILE
2972     *fp144;
2973     FILE
2974     *fp145;
2975     FILE
2976     *fp146;
2977     FILE
2978     *fp147;
2979     FILE
2980     *fp148;
2981     FILE
2982     *fp149;
2983     FILE
2984     *fp150;
2985     FILE
2986     *fp151;
2987     FILE
2988     *fp152;
2989     FILE
2990     *fp153;
2991     FILE
2992     *fp154;
2993     FILE
2994     *fp155;
2995     FILE
2996     *fp156;
2997     FILE
2998     *fp157;
2999     FILE
3000     *fp158;
3001     FILE
3002     *fp159;
3003     FILE
3004     *fp160;
3005     FILE
3006     *fp161;
3007     FILE
3008     *fp162;
3009     FILE
3010     *fp163;
3011     FILE
3012     *fp164;
3013     FILE
3014     *fp165;
3015     FILE
3016     *fp166;
3017     FILE
3018     *fp167;
3019     FILE
3020     *fp168;
3021     FILE
3022     *fp169;
3023     FILE
3024     *fp170;
3025     FILE
3026     *fp171;
3027     FILE
3028     *fp172;
3029     FILE
3030     *fp173;
3031     FILE
3032     *fp174;
3033     FILE
3034     *fp175;
3035     FILE
3036     *fp176;
3037     FILE
3038     *fp177;
3039     FILE
3040     *fp178;
3041     FILE
3042     *fp179;
3043     FILE
3044     *fp180;
3045     FILE
3046     *fp181;
3047     FILE
3048     *fp182;
3049     FILE
3050     *fp183;
3051     FILE
3052     *fp184;
3053     FILE
3054     *fp185;
3055     FILE
3056     *fp186;
3057     FILE
3058     *fp187;
3059     FILE
3060     *fp188;
3061     FILE
3062     *fp189;
3063     FILE
3064     *fp190;
3065     FILE
3066     *fp191;
3067     FILE
3068     *fp192;
3069     FILE
3070     *fp193;
3071     FILE
3072     *fp194;
3073     FILE
3074     *fp195;
3075     FILE
3076     *fp196;
3077     FILE
3078     *fp197;
3079     FILE
3080     *fp198;
3081     FILE
3082     *fp199;
3083     FILE
3084     *fp200;
3085     FILE
3086     *fp201;
3087     FILE
3088     *fp202;
3089     FILE
3090     *fp203;
3091     FILE
3092     *fp204;
3093     FILE
3094     *fp205;
3095     FILE
3096     *fp206;
3097     FILE
3098     *fp207;
3099     FILE
3100     *fp208;
3101     FILE
3102     *fp209;
3103     FILE
3104     *fp210;
3105     FILE
3106     *fp211;
3107     FILE
3108     *fp212;
3109     FILE
3110     *fp213;
3111     FILE
3112     *fp214;
3113     FILE
3114     *fp215;
3115     FILE
3116     *fp216;
3117     FILE
3118     *fp217;
3119     FILE
3120     *fp218;
3121     FILE
3122     *fp219;
3123     FILE
3124     *fp220;
3125     FILE
3126     *fp221;
3127     FILE
3128     *fp222;
3129     FILE
3130     *fp223;
3131     FILE
3132     *fp224;
3133     FILE
3134     *fp225;
3135     FILE
3136     *fp226;
3137     FILE
3138     *fp227;
3139     FILE
3140     *fp228;
3141     FILE
3142     *fp229;
3143     FILE
3144     *fp230;
3145     FILE
3146     *fp231;
3147     FILE
3148     *fp232;
3149     FILE
3150     *fp233;
3151     FILE
3152     *fp234;
3153     FILE
3154     *fp235;
3155     FILE
3156     *fp236;
3157     FILE
3158     *fp237;
3159     FILE
3160     *fp238;
3161     FILE
3162     *fp239;
3163     FILE
3164     *fp240;
3165     FILE
3166     *fp241;
3167     FILE
3168     *fp242;
3169     FILE
3170     *fp243;
3171     FILE
3172     *fp244;
3173     FILE
3174     *fp245;
3175     FILE
3176     *fp246;
3177     FILE
3178     *fp247;
3179     FILE
3180     *fp248;
3181     FILE
3182     *fp249;
3183     FILE
3184     *fp250;
3185     FILE
3186     *fp251;
3187     FILE
3188     *fp252;
3189     FILE
3190     *fp253;
3191     FILE
3192     *fp254;
3193     FILE
3194     *fp255;
3195     FILE
3196     *fp256;
3197     FILE
3198     *fp257;
3199     FILE
3200     *fp258;
3201     FILE
3202     *fp259;
3203     FILE
3204     *fp260;
3205     FILE
3206     *fp261;
3207     FILE
3208     *fp262;
3209     FILE
3210     *fp263;
3211     FILE
3212     *fp264;
3213     FILE
3214     *fp265;
3215     FILE
3216     *fp266;
3217     FILE
3218     *fp267;
3219     FILE
3220     *fp268;
3221     FILE
3222     *fp269;
3223     FILE
3224     *fp270;
3225     FILE
3226     *fp271;
3227     FILE
3228     *fp272;
3229     FILE
3230     *fp273;
3231     FILE
3232     *fp274;
3233     FILE
3234     *fp275;
3235     FILE
3236     *fp276;
3237     FILE
3238     *fp277;
3239     FILE
3240     *fp278;
3241     FILE
3242     *fp279;
3243     FILE
3244     *fp280;
3245     FILE
3246     *fp281;
3247     FILE
3248     *fp282;
3249     FILE
3250     *fp283;
3251     FILE
3252     *fp284;
3253     FILE
3254     *fp285;
3255     FILE
3256     *fp286;
3257     FILE
3258     *fp287;
3259     FILE
3260     *fp288;
3261     FILE
3262     *fp289;
3263     FILE
3264     *fp290;
3265     FILE
3266     *fp291;
3267     FILE
3268     *fp292;
3269     FILE
3270     *fp293;
3271     FILE
3272     *fp294;
3273     FILE
3274     *fp295;
3275     FILE
3276     *fp296;
3277     FILE
3278     *fp297;
3279     FILE
3280     *fp298;
3281     FILE
3282     *fp299;
3283     FILE
3284     *fp300;
3285     FILE
3286     *fp301;
3287     FILE
3288     *fp302;
3289     FILE
3290     *fp303;
3291     FILE
3292     *fp304;
3293     FILE
3294     *fp305;
3295     FILE
3296     *fp306;
3297     FILE
3298     *fp307;
3299     FILE
3300     *fp308;
3301     FILE
3302     *fp309;
3303     FILE
3304     *fp310;
3305     FILE
3306     *fp311;
3307     FILE
3308     *fp312;
3309     FILE
3310     *fp313;
3311     FILE
3312     *fp314;
3313     FILE
3314     *fp315;
3315     FILE
3316     *fp316;
3317     FILE
3318     *fp317;
3319     FILE
3320     *fp318;
3321     FILE
3322     *fp319;
3323     FILE
3324     *fp320;
3325     FILE
3326     *fp321;
3327     FILE
3328     *fp322;
3329     FILE
3330     *fp323;
3331     FILE
3332     *fp324;
3333     FILE
3334     *fp325;
3335     FILE
3336     *fp326;
3337     FILE
3338     *fp327;
3339     FILE
3340     *fp328;
3341     FILE
3342     *fp329;
3343     FILE
3344     *fp330;
3345     FILE
3346     *fp331;
3347     FILE
3348     *fp332;
3349     FILE
3350     *fp333;
3351     FILE
3352     *fp334;
3353     FILE
3354     *fp335;
3355     FILE
3356     *fp336;
3357     FILE
3358     *fp337;
3359     FILE
3360     *fp338;
3361     FILE
3362     *fp339;
3363     FILE
3364     *fp340;
3365     FILE
3366     *fp341;
3367     FILE
3368     *fp342;
3369     FILE
3370     *fp343;
3371     FILE
3372     *fp344;
3373     FILE
3374     *fp345;
3375     FILE
3376     *fp346;
3377     FILE
3378     *fp347;
3379     FILE
3380     *fp348;
3381     FILE
3382     *fp349;
3383     FILE
3384     *fp350;
3385     FILE
3386     *fp351;
3387     FILE
3388     *fp352;
3389     FILE
3390     *fp353;
3391     FILE
3392     *fp354;
3393     FILE
3394     *fp355;
3395     FILE
3396     *fp356;
3397     FILE
3398     *fp357;
3399     FILE
3400     *fp358;
3401     FILE
3402     *fp359;
3403     FILE
3404     *fp360;
3405     FILE
3406     *fp361;
3407     FILE
3408     *fp362;
3409     FILE
3410     *fp363;
3411     FILE
3412     *fp364;
3413     FILE
3414     *fp365;
3415     FILE
3416     *fp366;
3417     FILE
3418     *fp367;
3419     FILE
3420     *fp368;
3421     FILE
3422     *fp369;
3423     FILE
3424     *fp370;
3425     FILE
3426     *fp371;
3427     FILE
3428     *fp372;
3429     FILE
3430     *fp373;
3431     FILE
3432     *fp374;
3433     FILE
3434     *fp375;
3435     FILE
3436     *fp376;
3437     FILE
3438     *fp377;
3439     FILE
3440     *fp378;
3441     FILE
3442     *fp379;
3443     FILE
3444     *fp380;
3445     FILE
3446     *fp381;
3447     FILE
3448     *fp382;
3449     FILE
3450     *fp383;
3451     FILE
3452     *fp384;
3453     FILE
3454     *fp385;
3455     FILE
3456     *fp386;
3457     FILE
3458     *fp387;
3459     FILE
3460     *fp388;
3461     FILE
3462     *fp389;
3463     FILE
3464     *fp390;
3465     FILE
3466     *fp391;
3467     FILE
3468     *fp392;
3469     FILE
3470     *fp393;
3471     FILE
3472     *fp394;
3473     FILE
3474     *fp395;
3475     FILE
3476     *fp396;
3477     FILE
3478     *fp397;
3479     FILE
3480     *fp398;
3481     FILE
3482     *fp399;
3483     FILE
3484     *fp400;
3485     FILE
3486     *fp401;
3487     FILE
3488     *fp402;
3489     FILE
3490     *fp403;
3491     FILE
3492     *fp404;
3493     FILE
3494     *fp405;
3495     FILE
3496     *fp406;
3497     FILE
3498     *fp407;
3499     FILE
3500     *fp408;
3501     FILE
3502     *fp409;
3503     FILE
3504     *fp410;
3505     FILE
3506     *fp411;
3507     FILE
3508     *fp412;
3509     FILE
3510     *fp413;
3511     FILE
3512     *fp414;
3513     FILE
3514     *fp415;
3515     FILE
3516     *fp416;
3517     FILE
3518     *fp417;
3519     FILE
3520     *fp418;
3521     FILE
3522     *fp419;
3523     FILE
3524     *fp420;
3525     FILE
3526     *fp421;
3527     FILE
3528     *fp422;
3529     FILE
3530     *fp423;
3531     FILE
3532     *fp424;
3533     FILE
3534     *fp425;
3535     FILE
3536     *fp426;
3537     FILE
3538     *fp427;
3539     FILE
3540     *fp428;
3541     FILE
3542     *fp429;
3543     FILE
3544     *fp430;
3545     FILE
3546     *fp431;
3547     FILE
3548     *fp432;
3549     FILE
3550     *fp433;
3551     FILE
3552     *fp434;
3553     FILE
3554     *fp435;
3555     FILE
3556     *fp436;
3557     FILE
3558     *fp437;
3559     FILE
3560     *fp438;
3561     FILE
3562     *fp439;
3563     FILE
3564     *fp440;
3565     FILE
3566     *fp441;
3567     FILE
3568     *fp442;
3569     FILE
3570     *fp443;
3571     FILE
3572     *fp444;
3573     FILE
3574     *fp445;
3575     FILE
3576     *fp446;
3577     FILE
3578     *fp447;
3579     FILE
3580     *fp448;
3581     FILE
3582     *fp449;
3583     FILE
3584     *fp450;
3585     FILE
3586     *fp451;
3587     FILE
3588     *fp452;
3589     FILE
3590     *fp453;
3591     FILE
3592     *fp454;
3593     FILE
3594     *fp455;
3595     FILE
3596     *fp456;
3597     FILE
3598     *fp457;
3599     FILE
3600     *fp458;
3601     FILE
3602     *fp459;
3603     FILE
3604     *fp460;
3605     FILE
3606     *fp461;
3607     FILE
3608     *fp462;
3609     FILE
3610     *fp463;
3611     FILE
3612     *fp464;
3613     FILE
3614     *fp465;
3615     FILE
3616     *fp466;
3617     FILE
3618     *fp467;
3619     FILE
3620     *fp468;
3621     FILE
3622     *fp469;
3623     FILE
3624     *fp470;
3625     FILE
3626     *fp471;
3627     FILE
3628     *fp472;
3629     FILE
3630     *fp473;
3631     FILE
3632     *fp474;
3633     FILE
3634     *fp475;
3635     FILE
3636     *fp476;
3637     FILE
3638     *fp477;
3639     FILE
3640     *fp478;
3641     FILE
3642     *fp479;
3643     FILE
3644     *fp480;
3645     FILE
3646     *fp481;
3647     FILE
3648     *fp482;
3649     FILE
3650     *fp483;
3651     FILE
3652     *fp484;
3653     FILE
3654     *fp485;
3655     FILE
3656     *fp486;
3657     FILE
3658     *fp487;
3659     FILE
3660     *fp488;
3661     FILE
3662     *fp489;
3663     FILE
3664     *fp490;
3665     FILE
3666     *fp491;
3667     FILE
3668     *fp492;
3669     FILE
3670     *fp493;
3671     FILE
3672     *fp494;
3673     FILE
3674     *fp495;
3675     FILE
3676     *fp496;
3677     FILE
3678     *fp497;
3679     FILE
3680     *fp498;
3681     FILE
3682     *fp499;
3683     FILE
3684     *fp500;
3685     FILE
3686     *fp501;
3687     FILE
3688     *fp502;
3689     FILE
3690     *fp503;
3691     FILE
3692     *fp504;
3693     FILE
3694     *fp505;
3695     FILE
3696     *fp506;
3697     FILE
3698     *fp507;
3699     FILE
3700     *fp508;
3701     FILE
3702     *fp509;
3703     FILE
3704     *fp510;
3705     FILE
3706     *fp511;
3707     FILE
3708     *fp512;
3709     FILE
3710     *fp513;
3711     FILE
3712     *fp514;
3713     FILE
3714     *fp515;
3715     FILE
3716     *fp516;
3717     FILE
3718     *fp517;
3719     FILE
3720     *fp518;
3721     FILE
3722     *fp519;
3723     FILE
3724     *fp520;
3725     FILE
3726     *fp521;
3727     FILE
3728     *fp522;
3729     FILE
3730     *fp523;
3731     FILE
3732     *fp524;
3733     FILE
3734     *fp525;
3735     FILE
3736     *fp526;
3737     FILE
3738     *fp527;
3739     FILE
3740     *fp528;
3741     FILE
3742     *fp529;
3743     FILE
3744     *fp530;
3745     FILE
3746     *fp531;
3747     FILE
3748     *fp532;
3749     FILE
3750     *fp533;
3751     FILE
3752     *fp534;
3753     FILE
3754     *fp535;
3755     FILE
3756     *fp536;
3757     FILE
3758     *fp537;
3759     FILE
3760     *fp538;
3761     FILE
3762     *fp539;
3763     FILE
3764     *fp540;
3765     FILE
3766     *fp541;
3767     FILE
3768     *fp542;
3769     FILE
3770     *fp543;
3771     FILE
3772     *fp544;
3773     FILE
3774     *fp545;
3775     FILE
3776     *fp546;
3777     FILE
3778     *fp547;
3779     FILE
3780     *fp548;
3781     FILE
3782     *fp549;
3783     FILE
3784     *fp550;
3785     FILE
3786     *fp551;
3787     FILE
3788     *fp552;
3789     FILE
3790     *fp553;
3791     FILE
3792     *fp554;
3793     FILE
3794     *fp555;
3795     FILE
3796     *fp556;
3797     FILE
3798     *fp557;
3799     FILE
3800     *fp558;
3801     FILE
3802     *fp559;
3803     FILE
3804     *fp560;
3805     FILE
3806     *fp561;
3807     FILE
3808     *fp562;
3809     FILE
3810     *fp563;
3811     FILE
3812     *fp564;
3813     FILE
3814     *fp565;
3815     FILE
3816     *fp566;
3817     FILE
3818     *fp567;
3819     FILE
3820     *fp568;
3821     FILE
3822     *fp569;
3823     FILE
3824     *fp570;
3825     FILE
3826     *fp571;
3827     FILE
3828     *fp572;
3829     FILE
3830     *fp573;
3831     FILE
3832     *fp574;
3833     FILE
3834     *fp575;
3835     FILE
3836     *fp576;
3837     FILE
3838     *fp577;
3839     FILE
3840     *fp578;
3841     FILE
3842     *fp579;
3843     FILE
3844     *fp580;
3845     FILE
3846     *fp581;
3847     FILE
3848     *fp582;
3849     FILE
3850     *fp583;
3851     FILE
3852     *fp584;
3853     FILE
3854     *fp585;
3855     FILE
3856     *fp586;
3857     FILE
3858     *fp587;
3859     FILE
3860     *fp588;
3861     FILE
3862     *fp589;
3863     FILE
3864     *fp590;
3865     FILE
3866     *fp591;
3867     FILE
3868     *fp592;
3869     FILE
3870     *fp593;
3871     FILE
3872     *fp594;
3873     FILE
3874     *fp595;
3875     FILE
3876     *fp596;
3877     FILE
3878     *fp597;
3879     FILE
3880     *fp598;
3881     FILE
3882     *fp599;
3883     FILE
3884     *fp600;
3885     FILE
3886     *fp601;
3887     FILE
3888     *fp602;
3889     FILE
3890     *fp603;
3891     FILE
3892     *fp604;
3893     FILE
3894     *fp605;
3895     FILE
3896     *fp606;
3897     FILE
3898     *fp607;
3899     FILE
3900     *fp608;
3901     FILE
3902     *fp609;
3903     FILE
3904     *fp610;
3905     FILE
3906     *fp611;
3907     FILE
3908     *fp612;
3909     FILE
3910     *fp613;
3911     FILE
3912     *fp614;
3913     FILE
3914     *fp615;
3915     FILE
3916     *fp616;
3917     FILE
3918     *fp617;
3919     FILE
3920     *fp618;
3921     FILE
3922     *fp619;
3923     FILE
3924     *fp620;
3925     FILE
3926     *fp621;
3927     FILE
3928     *fp622;
3929     FILE
3930     *fp623;
3931     FILE
3932     *fp624;
3933     FILE
3934     *fp625;
3935     FILE
3936     *fp626;
3937     FILE
3938     *fp627;
3939     FILE
3940     *fp628;
3941     FILE
3942     *fp629;
3943     FILE
3944     *fp630;
3945     FILE
3946
```



```

2838 2      *exitd = 0;
2839 2      *demo_exit_status = 0;
2840 2      *skiping_leading_whitespace = FALSE;
2841 2      n = 0;
2842 2      }
2843 2      else
2844 2      {
2845 2          n = *parsepos;
2846 2      }
2847 1      while (!prototated && !done)
2848 1      {
2849 1          int start_ec;
2850 1          int r;
2851 2
2852 2
2853 2
2854 2      /*
2855 2      * wait for the next character from the remote
2856 2      * read stream ignores interrupts, unless
2857 2      * they were due to the attention signal.
2858 2      */
2859 2
2860 2      start_ec = atn_ec;
2861 2      do
2862 2      {
2863 2          r = fd_await_test_intel(remote_fd);
2864 2          if (0 == r)
2865 2          {
2866 2              return ret_status;
2867 2          }
2868 2          while (r == -1 && errno == EINTR && atn_ec == start_ec);
2869 2
2870 2      /*
2871 2      * now that there is a character, read it
2872 2      */
2873 2      c = 0;
2874 2
2875 2      if (r != -1)
2876 2      {
2877 2          do
2878 2          {
2879 2              r = Ctl_read(remote_fd, &c, 1, 0);
2880 2              while((-1 == r) && (errno == EINTR) && (
2881 2                  atn_ec == start_ec));
2882 2          }
2883 2          if (r != 1)
2884 2          {
2885 2              *exitd = SPEXIT_REMOTE_STREAM_FAIL;
2886 2              prototated = 1;
2887 2              ret_status = TRUE;
2888 2              break;
2889 2          }
2890 2
2891 2          previous_state = *state_ptr;
2892 2          *state_ptr = "next_state_ptr";
2893 2
2894 2          switch (*state_ptr)
2895 2          {
2896 2              case INSTATE_SEARCH_PREFIX:
2897 2                  if (debugmode)
2898 2                  {
2899 2                      rbe_log_state(0, "AUXPROC: (
2900 2                          %(c) INSTATE_SEARCH_PREFIX\n", c);
2901 2                  }

```

```

2901 3      {
2902 4          if (c == REMOVED_MAGIC_PREFIX(0))
2903 4          {
2904 4              *next_state_ptr = INSTATE_SEARCH_PREFIX;
2905 4              n = 0;
2906 4              *parsepos = 1;
2907 4          }
2908 4          break;
2909 3      }
2910 3      case INSTATE_SEARCH_PREFIX:
2911 3      {
2912 3          if (debugmode)
2913 3          {
2914 3              rbe_log_state(0, "AUXPROC: (
2915 3                  %(c) INSTATE_SEARCH_PREFIX\n", c);
2916 3          }
2917 3
2918 3          if (c != REMOVED_MAGIC_PREFIX(0))
2919 3          {
2920 3              *next_state_ptr = INSTATE_SEARCH_PREFIX;
2921 3          }
2922 3          else
2923 3          {
2924 3              n++;
2925 3              if (*parsepos++
2926 3                  < REMOVED_MAGIC_LENGTH)
2927 3              {
2928 3                  *next_state_ptr = INSTATE_GATHER_COOKIE;
2929 3                  n = 0;
2930 3                  *parsepos = 0;
2931 3              }
2932 3              break;
2933 3          }
2934 3      }
2935 3      case INSTATE_GATHER_COOKIE:
2936 3      {
2937 3          if (debugmode)
2938 3          {
2939 3              rbe_log_state(0, "AUXPROC: (
2940 3                  %(c) INSTATE_GATHER_COOKIE\n", c);
2941 3          }
2942 3
2943 3          if (c == REMOVED_MAGIC_SUFFIX(0))
2944 3          {
2945 3              /*
2946 3              * Got the cookie
2947 3              */
2948 3              cookiebuf[n] = '\0';
2949 3              next_state_ptr = INSTATE_SEARCH_SUFFIX;
2950 3              n = 1;
2951 3              *parsepos = 1;
2952 3          }
2953 3          else if (isctrl(REMOVED_COOKIE_CHARS, c) == NULL)
2954 3          {
2955 3              /*
2956 3              * We found a valid prefix (else we would
2957 3              * not be here) but the cookie contains an
2958 3              * illegal character. Should not happen.
2959 3              * There has been some protocol failure.
2960 3              */
2961 3              *exitd = SPEXIT_REMOTE_STREAM_FAIL;
2962 3              ret_status = TRUE;
2963 3              prototated = 1;
2964 3          }
2965 3          else if (n == REMOVED_MAX_COOKIE_LEN)
2966 3          {
2967 3              *exitd = SPEXIT_MAX_COOKIE_LEN;
2968 3          }

```

```

2985 4 /*
2986 4  * Should have seen the SUFFIX(0) by now.
2987 4  */
2988 4
2989 4
2990 4 *exitp = SPEXIT_REMOTE_STDBRR_PROTOCOL;
2991 4 rfc_status = TRUE;
2992 4 protected = 1;
2993 4 }
2994 4 else
2995 4 {
2996 4     /*
2997 4     * another cookie character.
2998 4     */
2999 4     cookiebuf[n] = c;
3000 4     n++;
3001 4     (*parsePos)++;
3002 4 }
3003 4 break;
3004 4
3005 4 case INSTSTATE_SEARCH_SUFFIX:
3006 4     if (debugmode)
3007 4     {
3008 4         rfc_log_status(0, "INSTSTATE_SEARCH_SUFFIX", c);
3009 4         /*
3010 4         * (c) INSTSTATE_SEARCH_SUFFIX, n", c);
3011 4         */
3012 4     }
3013 4     if (c != REMPD_MAGIC_SUFFIX(n))
3014 4     {
3015 4         /*
3016 4         * exitp = SPEXIT_REMOTE_STDBRR_PROTOCOL;
3017 4         * rfc_status = TRUE;
3018 4         * protected = 1;
3019 4         */
3020 4     }
3021 4     else
3022 4     {
3023 4         n++;
3024 4         (*parsePos)++;
3025 4         if (n == REMPD_MAGIC_LENGTH)
3026 4         {
3027 4             /*
3028 4             * next_state_ptr = INSTSTATE_NEWLINE;
3029 4             */
3030 4             break;
3031 4         }
3032 4         case INSTSTATE_GATHER_STATUS:
3033 4             if (debugmode)
3034 4             {
3035 4                 rfc_log_status(0, "INSTSTATE_GATHER_STATUS", c);
3036 4                 /*
3037 4                 * (c) INSTSTATE_GATHER_STATUS, n", c);
3038 4                 */
3039 4             }
3040 4             if (!isdigit(c))
3041 4             {
3042 4                 char xc[12];
3043 4                 xc[0] = c;
3044 4                 xc[1] = '\0';
3045 4                 temp_exit_status = 10;
3046 4                 temp_exit_status += atoi(xc);
3047 4                 n++;
3048 4                 (*parsePos)++;
3049 4                 *skipping_leading_whitespace = FALSE;
3050 4             }
3051 4             else if (c == '\n')
3052 4             {
3053 4                 }
3054 4             }
3055 4             }
3056 4             }
3057 4             }
3058 4             }
3059 4             }
3060 4             }
3061 4             }
3062 4             }
3063 4             }
3064 4             }
3065 4             }
3066 4             }
3067 4             }
3068 4             }
3069 4             }
3070 4             }
3071 4             }
3072 4             }
3073 4             }
3074 4             }
3075 4             }
3076 4             }
3077 4             }
3078 4             }
3079 4             }
3080 4             }
3081 4             }
3082 4             }
3083 4             }
3084 4             }
3085 4             }
3086 4             }
3087 4             }
3088 4             }
3089 4             }
3090 4             }
3091 4             }
3092 4             }
3093 4             }
3094 4             }
3095 4             }
3096 4             }
3097 4             }
3098 4             }
3099 4             }
3100 4             }
3101 4             }
3102 4             }
3103 4             }
3104 4             }
3105 4             }
3106 4             }
3107 4             }
3108 4             }
3109 4             }
3110 4             }
3111 4             }
3112 4             }
3113 4             }
3114 4             }
3115 4             }
3116 4             }
3117 4             }
3118 4             }
3119 4             }
3120 4             }
3121 4             }
3122 4             }
3123 4             }
3124 4             }
3125 4             }
3126 4             }
3127 4             }
3128 4             }
3129 4             }
3130 4             }
3131 4             }
3132 4             }
3133 4             }
3134 4             }
3135 4             }
3136 4             }
3137 4             }
3138 4             }
3139 4             }
3140 4             }
3141 4             }
3142 4             }
3143 4             }
3144 4             }
3145 4             }
3146 4             }
3147 4             }
3148 4             }
3149 4             }
3150 4             }
3151 4             }
3152 4             }
3153 4             }
3154 4             }
3155 4             }
3156 4             }
3157 4             }
3158 4             }
3159 4             }
3160 4             }
3161 4             }
3162 4             }
3163 4             }
3164 4             }
3165 4             }
3166 4             }
3167 4             }
3168 4             }
3169 4             }
3170 4             }
3171 4             }
3172 4             }
3173 4             }
3174 4             }
3175 4             }
3176 4             }
3177 4             }
3178 4             }
3179 4             }
3180 4             }
3181 4             }
3182 4             }
3183 4             }
3184 4             }
3185 4             }
3186 4             }
3187 4             }
3188 4             }
3189 4             }
3190 4             }
3191 4             }
3192 4             }
3193 4             }
3194 4             }
3195 4             }
3196 4             }
3197 4             }
3198 4             }
3199 4             }
3200 4             }
3201 4             }
3202 4             }
3203 4             }
3204 4             }
3205 4             }
3206 4             }
3207 4             }
3208 4             }
3209 4             }
3210 4             }
3211 4             }
3212 4             }
3213 4             }
3214 4             }
3215 4             }
3216 4             }
3217 4             }
3218 4             }
3219 4             }
3220 4             }
3221 4             }
3222 4             }
3223 4             }
3224 4             }
3225 4             }
3226 4             }
3227 4             }
3228 4             }
3229 4             }
3230 4             }
3231 4             }
3232 4             }
3233 4             }
3234 4             }
3235 4             }
3236 4             }
3237 4             }
3238 4             }
3239 4             }
3240 4             }
3241 4             }
3242 4             }
3243 4             }
3244 4             }
3245 4             }
3246 4             }
3247 4             }
3248 4             }
3249 4             }
3250 4             }
3251 4             }
3252 4             }
3253 4             }
3254 4             }
3255 4             }
3256 4             }
3257 4             }
3258 4             }
3259 4             }
3260 4             }
3261 4             }
3262 4             }
3263 4             }
3264 4             }
3265 4             }
3266 4             }
3267 4             }
3268 4             }
3269 4             }
3270 4             }
3271 4             }
3272 4             }
3273 4             }
3274 4             }
3275 4             }
3276 4             }
3277 4             }
3278 4             }
3279 4             }
3280 4             }
3281 4             }
3282 4             }
3283 4             }
3284 4             }
3285 4             }
3286 4             }
3287 4             }
3288 4             }
3289 4             }
3290 4             }
3291 4             }
3292 4             }
3293 4             }
3294 4             }
3295 4             }
3296 4             }
3297 4             }
3298 4             }
3299 4             }
3300 4             }
3301 4             }
3302 4             }
3303 4             }
3304 4             }
3305 4             }
3306 4             }
3307 4             }
3308 4             }
3309 4             }
3310 4             }
3311 4             }
3312 4             }
3313 4             }
3314 4             }
3315 4             }
3316 4             }
3317 4             }
3318 4             }
3319 4             }
3320 4             }
3321 4             }
3322 4             }
3323 4             }
3324 4             }
3325 4             }
3326 4             }
3327 4             }
3328 4             }
3329 4             }
3330 4             }
3331 4             }
3332 4             }
3333 4             }
3334 4             }
3335 4             }
3336 4             }
3337 4             }
3338 4             }
3339 4             }
3340 4             }
3341 4             }
3342 4             }
3343 4             }
3344 4             }
3345 4             }
3346 4             }
3347 4             }
3348 4             }
3349 4             }
3350 4             }
3351 4             }
3352 4             }
3353 4             }
3354 4             }
3355 4             }
3356 4             }
3357 4             }
3358 4             }
3359 4             }
3360 4             }
3361 4             }
3362 4             }
3363 4             }
3364 4             }
3365 4             }
3366 4             }
3367 4             }
3368 4             }
3369 4             }
3370 4             }
3371 4             }
3372 4             }
3373 4             }
3374 4             }
3375 4             }
3376 4             }
3377 4             }
3378 4             }
3379 4             }
3380 4             }
3381 4             }
3382 4             }
3383 4             }
3384 4             }
3385 4             }
3386 4             }
3387 4             }
3388 4             }
3389 4             }
3390 4             }
3391 4             }
3392 4             }
3393 4             }
3394 4             }
3395 4             }
3396 4             }
3397 4             }
3398 4             }
3399 4             }
3400 4             }
3401 4             }
3402 4             }
3403 4             }
3404 4             }
3405 4             }
3406 4             }
3407 4             }
3408 4             }
3409 4             }
3410 4             }
3411 4             }
3412 4             }
3413 4             }
3414 4             }
3415 4             }
3416 4             }
3417 4             }
3418 4             }
3419 4             }
3420 4             }
3421 4             }
3422 4             }
3423 4             }
3424 4             }
3425 4             }
3426 4             }
3427 4             }
3428 4             }
3429 4             }
3430 4             }
3431 4             }
3432 4             }
3433 4             }
3434 4             }
3435 4             }
3436 4             }
3437 4             }
3438 4             }
3439 4             }
3440 4             }
3441 4             }
3442 4             }
3443 4             }
3444 4             }
3445 4             }
3446 4             }
3447 4             }
3448 4             }
3449 4             }
3450 4             }
3451 4             }
3452 4             }
3453 4             }
3454 4             }
3455 4             }
3456 4             }
3457 4             }
3458 4             }
3459 4             }
3460 4             }
3461 4             }
3462 4             }
3463 4             }
3464 4             }
3465 4             }
3466 4             }
3467 4             }
3468 4             }
3469 4             }
3470 4             }
3471 4             }
3472 4             }
3473 4             }
3474 4             }
3475 4             }
3476 4             }
3477 4             }
3478 4             }
3479 4             }
3480 4             }
3481 4             }
3482 4             }
3483 4             }
3484 4             }
3485 4             }
3486 4             }
3487 4             }
3488 4             }
3489 4             }
3490 4             }
3491 4             }
3492 4             }
3493 4             }
3494 4             }
3495 4             }
3496 4             }
3497 4             }
3498 4             }
3499 4             }
3500 4             }
3501 4             }
3502 4             }
3503 4             }
3504 4             }
3505 4             }
3506 4             }
3507 4             }
3508 4             }
3509 4             }
3510 4             }
3511 4             }
3512 4             }
3513 4             }
3514 4             }
3515 4             }
3516 4             }
3517 4             }
3518 4             }
3519 4             }
3520 4             }
3521 4             }
3522 4             }
3523 4             }
3524 4             }
3525 4             }
3526 4             }
3527 4             }
3528 4             }
3529 4             }
3530 4             }
3531 4             }
3532 4             }
3533 4             }
3534 4             }
3535 4             }
3536 4             }
3537 4             }
3538 4             }
3539 4             }
3540 4             }
3541 4             }
3542 4             }
3543 4             }
3544 4             }
3545 4             }
3546 4             }
3547 4             }
3548 4             }
3549 4             }
3550 4             }
3551 4             }
3552 4             }
3553 4             }
3554 4             }
3555 4             }
3556 4             }
3557 4             }
3558 4             }
3559 4             }
3560 4             }
3561 4             }
3562 4             }
3563 4             }
3564 4             }
3565 4             }
3566 4             }
3567 4             }
3568 4             }
3569 4             }
3570 4             }
3571 4             }
3572 4             }
3573 4             }
3574 4             }
3575 4             }
3576 4             }
3577 4             }
3578 4             }
3579 4             }
3580 4             }
3581 4             }
3582 4             }
3583 4             }
3584 4             }
3585 4             }
3586 4             }
3587 4             }
3588 4             }
3589 4             }
3590 4             }
3591 4             }
3592 4             }
3593 4             }
3594 4             }
3595 4             }
3596 4             }
3597 4             }
3598 4             }
3599 4             }
3600 4             }
3601 4             }
3602 4             }
3603 4             }
3604 4             }
3605 4             }
3606 4             }
3607 4             }
3608 4             }
3609 4             }
3610 4             }
3611 4             }
3612 4             }
3613 4             }
3614 4             }
3615 4             }
3616 4             }
3617 4             }
3618 4             }
3619 4             }
3620 4             }
3621 4             }
3622 4             }
3623 4             }
3624 4             }
3625 4             }
3626 4             }
3627 4             }
3628 4             }
3629 4             }
3630 4             }
3631 4             }
3632 4             }
3633 4             }
3634 4             }
3635 4             }
3636 4             }
3637 4             }
3638 4             }
3639 4             }
3640 4             }
3641 4             }
3642 4             }
3643 4             }
3644 4             }
3645 4             }
3646 4             }
3647 4             }
3648 4             }
3649 4             }
3650 4             }
3651 4             }
3652 4             }
3653 4             }
3654 4             }
3655 4             }
3656 4             }
3657 4             }
3658 4             }
3659 4             }
3660 4             }
3661 4             }
3662 4             }
3663 4             }
3664 4             }
3665 4             }
3666 4             }
3667 4             }
3668 4             }
3669 4             }
3670 4             }
3671 4             }
3672 4             }
3673 4             }
3674 4             }
3675 4             }
3676 4             }
3677 4             }
3678 4             }
3679 4             }
3680 4             }
3681 4             }
3682 4             }
3683 4             }
3684 4             }
3685 4             }
3686 4             }
3687 4             }
3688 4             }
3689 4             }
3690 4             }
3691 4             }
3692 4             }
3693 4             }
3694 4             }
3695 4             }
3696 4             }
3697 4             }
3698 4             }
3699 4             }
3700 4             }
3701 4             }
3702 4             }
3703 4             }
3704 4             }
3705 4             }
3706 4             }
3707 4             }
3708 4             }
3709 4             }
3710 4             }
3711 4             }
3712 4             }
3713 4             }
3714 4             }
3715 4             }
3716 4             }
3717 4             }
3718 4             }
3719 4             }
3720 4             }
3721 4             }
3722 4             }
3723 4             }
3724 4             }
3725 4             }
3726 4             }
3727 4             }
3728 4             }
3729 4             }
3730 4             }
3731 4             }
3732 4             }
3733 4             }
3734 4             }
3735 4             }
3736 4             }
3737 4             }
3738 4             }
3739 4             }
3740 4             }
3741 4             }
3742 4             }
3743 4             }
3744 4             }
3745 4             }
3746 4             }
3747 4             }
3748 4             }
3749 4             }
3750 4             }
3751 4             }
3752 4             }
3753 4             }
3754 4             }
3755 4             }
3756 4             }
3757 4             }
3758 4             }
3759 4             }
3760 4             }
3761 4             }
3762 4             }
3763 4             }
3764 4             }
3765 4             }
3766 4             }
3767 4             }
3768 4             }
3769 4             }
3770 4             }
3771 4             }
3772 4             }
3773 4             }
3774 4             }
3775 4             }
3776 4             }
3777 4             }
3778 4             }
3779 4             }
3780 4             }
3781 4             }
3782 4             }
3783 4             }
3784 4             }
3785 4             }
3786 4             }
3787 4             }
3788 4             }
3789 4             }
3790 4             }
3791 4             }
3792 4             }
3793 4             }
3794 4             }
3795 4             }
3796 4             }
3797 4             }
3798 4             }
3799 4             }
3800 4             }
3801 4             }
3802 4             }
3803 4             }
3804 4             }
3805 4             }
3806 4             }
3807 4             }
3808 4             }
3809 4             }
3810 4             }
3811 4             }
3812 4             }
3813 4             }
3814 4             }
3815 4             }
3816 4             }
3817 4             }
3818 4             }
3819 4             }
3820 4             }
3821 4             }
3822 4             }
3823 4             }
3824 4             }
3825 4             }
3826 4             }
3827 4             }
3828 4             }
3829 4             }
3830 4             }
3831 4             }
3832 4             }
3833 4             }
3834 4             }
3835 4             }
3836 4             }
3837 4             }
3838 4             }
3839 4             }
3840 4             }
3841 4             }
3842 4             }
3843 4             }
3844 4             }
3845 4             }
3846 4             }
3847 4             }
3848 4             }
3849 4             }
3850 4             }
3851 4             }
3852 4             }
3853 4             }
3854 4             }
3855 4             }
3856 4             }
3857 4             }
3858 4             }
3859 4             }
3860 4             }
3861 4             }
3862 4             }
3863 4             }
3864 4             }
3865 4             }
3866 4             }
3867 4             }
3868 4             }
3869 4             }
3870 4             }
3871 4             }
3872 4             }
3873 4             }
3874 4             }
3875 4             }
3876 4             }
3877 4             }
3878 4             }
3879 4             }
3880 4             }
3881 4             }
3882 4             }
3883 4             }
3884 4             }
3885 4             }
3886 4             }
3887 4             }
3888 4             }
3889 4             }
3890 4             }
3891 4             }
3892 4             }
3893 4             }
3894 4             }
3895 4             }
3896 4             }
3897 4             }
3898 4             }
3899 4             }
3900 4             }
3901 4             }
3902 4             }
3903 4             }
3904 4             }
3905 4             }
3906 4             }
3907 4             }
3908 4             }
3909 4             }
3910 4             }
3911 4             }
3912 4             }
3913 4             }
3914 4             }
3915 4             }
3916 4             }
3917 4             }
3918 4             }
3919 4             }
3920 4             }
3921 4             }
3922 4             }
3923 4             }
3924 4             }
3925 4             }
3926 4             }
3927 4             }
3928 4             }
3929 4             }
3930 4             }
3931 4             }
3932 4             }
3933 4             }
3934 4             }
3935 4             }
3936 4             }
3937 4             }
3938 4             }
3939 4             }
3940 4             }
3941 4             }
3942 4             }
3943 4             }
3944 4             }
3945 4             }
3946 4             }
3947 4             }
3948 4             }
3949 4             }
3950 4             }
3951 4             }
3952 4             }
3953 4             }
3954 4             }
3955 4             }
3956 4             }
3957 4             }
3958 4             }
3959 4             }
3960 4             }
3961 4             }
3962 4             }
3963 4             }
3964 4             }
3965 4             }
3966 4             }
3967 4             }
3968 4             }
3969 4             }
3970 4             }
3971 4             }
3972 4             }
3973 4             }
3974 4             }
3975 4             }
3976 4             }
3977 4             }
3978 4             }
3979 4             }
3980 4             }
3981 4             }
3982 4             }
3983 4             }
3984 4             }
3985 4             }
3986 4             }
3987 4             }
3988 4             }
3989 4             }
3990 4             }
3991 4             }
3992 4             }
3993 4             }
3994 4             }
3995 4             }
3996 4             }
3997 4             }
3998 4             }
3999 4             }
4000 4             }

```

```

3018 4 {
3019 4     *exitp = temp_exit_status;
3020 4     rfc_status = TRUE;
3021 4     done = 1;
3022 4 }
3023 4 else if (c == ' ' &&
3024 4     {n == 0} || *skipping_leading_whitespace) )
3025 4 {
3026 4     *skipping_leading_whitespace = TRUE;
3027 4 }
3028 4 /*
3029 4 * no-op, skip leading whitespace
3030 4 */
3031 4 }
3032 4 else
3033 4 {
3034 4     *exitp = SPEXIT_REMOTE_STDBRR_PROTOCOL;
3035 4     rfc_status = TRUE;
3036 4     protected = 1;
3037 4 }
3038 4 break;
3039 4
3040 4 case INSTSTATE_COPY_NO_STOUT:
3041 4     if (debugmode)
3042 4     {
3043 4         /*
3044 4         * Eventually, the protocol should include
3045 4         * an explicit length for this state, so will not
3046 4         * we can do a large read/write based on gunk being
3047 4         * copied to stdout.
3048 4         * For now, just shove the characters at stdout
3049 4         * and stop as soon as we fix PREFIX(0)
3050 4         */
3051 4         if (c != REMPD_MAGIC_PREFIX(0))
3052 4         {
3053 4             /*
3054 4             * build a buffer to write to the restore_log
3055 4             */
3056 4             if (('n' == c) || ((*msgPos) > MSGBUFSIZE))
3057 4             {
3058 4                 msgLogBuf["msgPos"] = '\0';
3059 4                 (void) rfc_log_status(0,
3060 4                     "as: %s",
3061 4                     rfc_restore_log,
3062 4                     msgLogBuf);
3063 4                 if (writeProg)
3064 4                 {
3065 4                     EDWRITEPROGMSG RESTORE_MCMD_UNKNOWN,
3066 4                         0,
3067 4                         msgLogBuf);
3068 4                 }
3069 4                 *msgPos = 0;
3070 4                 *msgPos = MSGBUFSIZE + 1;
3071 4                 *msgPos = 0;
3072 4                 *msgPos = 0;
3073 4                 *msgPos = 0;
3074 4                 *msgPos = 0;
3075 4                 *msgPos = 0;
3076 4                 *msgPos = 0;
3077 4                 *msgPos = 0;
3078 4                 *msgPos = 0;
3079 4                 *msgPos = 0;
3080 4                 *msgPos = 0;
3081 4                 *msgPos = 0;
3082 4                 *msgPos = 0;
3083 4                 *msgPos = 0;
3084 4                 *msgPos = 0;
3085 4                 *msgPos = 0;
3086 4                 *msgPos = 0;
3087 4                 *msgPos = 0;
3088 4                 *msgPos = 0;
3089 4                 *msgPos = 0;
3090 4                 *msgPos = 0;
3091 4                 *msgPos = 0;
3092 4                 *msgPos = 0;
3093 4                 *msgPos = 0;
3094 4                 *msgPos = 0;
3095 4                 *msgPos = 0;
3096 4                 *msgPos = 0;
3097 4                 *msgPos = 0;
3098 4                 *msgPos = 0;
3099 4                 *msgPos = 0;
3100 4                 *msgPos = 0;
3101 4                 *msgPos = 0;
3102 4                 *msgPos = 0;
3103 4                 *msgPos = 0;
3104 4                 *msgPos = 0;
3105 4                 *msgPos = 0;
3106 4                 *msgPos = 0;
3107 4                 *msgPos = 0;
3108 4                 *msgPos = 0;
3109 4                 *msgPos = 0;
3110 4                 *msgPos = 0;
3111 4                 *msgPos = 0;
3112 4                 *msgPos = 0;
3113 4                 *msgPos = 0;
3114 4                 *msgPos = 0;
3115 4                 *msgPos = 0;
3116 4                 *msgPos = 0;
3117 4                 *msgPos = 0;
3118 4                 *msgPos = 0;
3119 4                 *msgPos = 0;
3120 4                 *msgPos = 0;
3121 4                 *msgPos = 0;
3122 4                 *msgPos = 0;
3123 4                 *msgPos = 0;
3124 4                 *msgPos = 0;
3125 4                 *msgPos = 0;
3126 4                 *msgPos = 0;
3127 4                 *msgPos = 0;
3128 4                 *msgPos = 0;
3129 4                 *msgPos = 0;
3130 4                 *msgPos = 0;
3131 4                 *msgPos = 0;
3132 4                 *msgPos = 0;
3133 4                 *msgPos = 0;
3134 4                 *msgPos = 0;
3135 4                 *msgPos = 0;
3136 4                 *msgPos = 0;
3137 4                 *msgPos = 0;
3138 4                 *msgPos = 0;
3139 4                 *msgPos = 0;
3140 4                 *msgPos = 0;
3141 4                 *msgPos = 0;
3142 4                 *msgPos = 0;
3143 4                 *msgPos = 0;
3144 4                 *msgPos = 0;
3145 4                 *msgPos = 0;
3146 4                 *msgPos = 0;
3147 4                 *msgPos = 0;
3148 4                 *msgPos = 0;
3149 4                 *msgPos = 0;
3150 4                 *msgPos = 0;
3151 4                 *msgPos = 0;
3152 4                 *msgPos = 0;
3153 4                 *msgPos = 0;
3154 4                 *msgPos = 0;
3155 4                 *msgPos = 0;
3156 4                 *msgPos = 0;
3157 4                 *msgPos = 0;
3158 4                 *msgPos = 0;
3159 4                 *msgPos = 0;
3160 4                 *msgPos = 0;
3161 4                 *msgPos = 0;
3162 4                 *msgPos = 0;
3163 4                 *msgPos = 0;
3164 4                 *msgPos = 0;
3165 4                 *msgPos = 0;
3166 4                 *msgPos = 0;
3167 4                 *msgPos = 0;
3168 4                 *msgPos = 0;
3169 4                 *msgPos = 0;
3170 4                 *msgPos = 0;
3171 4                 *msgPos = 0;
3172 4                 *msgPos = 0;
3173 4                 *msgPos = 0;
3174 4                 *msgPos = 0;
3175 4                 *msgPos = 0;
3176 4                 *msgPos = 0;
3177 4                 *msgPos = 0;
3178 4                 *msgPos = 0;
3179 4                 *msgPos = 0;
3180 4                 *msgPos = 0;
3181 4                 *msgPos = 0;
3182 4                 *msgPos = 0;
3183 4                 *msgPos = 0;
3184 4                 *msgPos = 0;
3185 4                 *msgPos = 0;
3186 4                 *msgPos = 0;
3187 4                 *msgPos = 0;
3188 4                 *msgPos = 0;
3189 4                 *msgPos = 0;
3190 4                 *msgPos = 0;
3191 4                 *msgPos = 0;
3192 4                 *msgPos = 0;
3193 4                 *msgPos = 0;
3194 4                 *msgPos = 0;
3195 4                
```



```

3135 1         }
3136 2         write_rpc = WriteMsg(&restore_msgline_prog_fd, xcpiogen_msg);
3137 3         msg_count++;
3138 4         free(xcpiogen_msg.pcm_body);
3139 5     }
3140 6     memset(&xcpiogen_msg, 0, sizeof(prog_chan_msg));
3141 7 }
3142 8 if (--fd_test)
3143 9 {
3144 10     (void) rbe_log_status(BARECOVER_MERR(errno),
3145 11                          "Encountered error testing xcpiogen file
3146 12                          descriptor.");
3147 13 }
3148 14 return -1;
3149 15 }
3150 16 return msg_count;
3151 17 }
3152 18
3153 19
3154 20
3155 21
3156 22
3157 23
3158 24
3159 25
3160 26
3161 27
3162 28
3163 29
3164 30
3165 31
3166 32
3167 33
3168 34
3169 35
3170 36
3171 37
3172 38
3173 39
3174 40
3175 41
3176 42
3177 43
3178 44
3179 45
3180 46
3181 47
3182 48
3183 49
3184 50
3185 51
3186 52
3187 53
3188 54
3189 55
3190 56
3191 57
3192 58
3193 59
3194 60
3195 61
3196 62
3197 63
3198 64
3199 65
3200 66
3201 67
3202 68
3203 69
3204 70
3205 71
3206 72
3207 73
3208 74
3209 75
3210 76
3211 77
3212 78
3213 79
3214 80
3215 81
3216 82
3217 83
3218 84
3219 85
3220 86
3221 87
3222 88
3223 89
3224 90
3225 91
3226 92
3227 93
3228 94
3229 95
3230 96
3231 97
3232 98
3233 99
3234 100
3235 101
3236 102
3237 103
3238 104
3239 105
3240 106
3241 107
3242 108
3243 109
3244 110
3245 111
3246 112
3247 113
3248 114
3249 115
3250 116
3251 117
3252 118
3253 119
3254 120
3255 121
3256 122
3257 123
3258 124
3259 125
3260 126
3261 127
3262 128
3263 129
3264 130
3265 131
3266 132
3267 133
3268 134
3269 135
3270 136
3271 137
3272 138
3273 139
3274 140
3275 141
3276 142
3277 143
3278 144
3279 145
3280 146
3281 147
3282 148
3283 149
3284 150
3285 151
3286 152
3287 153
3288 154
3289 155
3290 156
3291 157
3292 158
3293 159
3294 160
3295 161
3296 162
3297 163
3298 164
3299 165
3300 166
3301 167
3302 168
3303 169
3304 170
3305 171
3306 172
3307 173
3308 174
3309 175
3310 176
3311 177
3312 178
3313 179
3314 180
3315 181
3316 182
3317 183
3318 184
3319 185
3320 186
3321 187
3322 188
3323 189
3324 190
3325 191
3326 192
3327 193
3328 194
3329 195
3330 196
3331 197
3332 198
3333 199
3334 200
3335 201
3336 202
3337 203
3338 204
3339 205
3340 206
3341 207
3342 208
3343 209
3344 210
3345 211
3346 212
3347 213
3348 214
3349 215
3350 216
3351 217
3352 218
3353 219
3354 220
3355 221
3356 222
3357 223
3358 224
3359 225
3360 226
3361 227
3362 228
3363 229
3364 230
3365 231
3366 232
3367 233
3368 234
3369 235
3370 236
3371 237
3372 238
3373 239
3374 240
3375 241
3376 242
3377 243
3378 244
3379 245
3380 246
3381 247
3382 248
3383 249
3384 250
3385 251
3386 252
3387 253
3388 254
3389 255
3390 256
3391 257
3392 258
3393 259
3394 260
3395 261
3396 262
3397 263
3398 264
3399 265
3400 266
3401 267
3402 268
3403 269
3404 270
3405 271
3406 272
3407 273
3408 274
3409 275
3410 276
3411 277
3412 278
3413 279
3414 280
3415 281
3416 282
3417 283
3418 284
3419 285
3420 286
3421 287
3422 288
3423 289
3424 290
3425 291
3426 292
3427 293
3428 294
3429 295
3430 296
3431 297
3432 298
3433 299
3434 300
3435 301
3436 302
3437 303
3438 304
3439 305
3440 306
3441 307
3442 308
3443 309
3444 310
3445 311
3446 312
3447 313
3448 314
3449 315
3450 316
3451 317
3452 318
3453 319
3454 320
3455 321
3456 322
3457 323
3458 324
3459 325
3460 326
3461 327
3462 328
3463 329
3464 330
3465 331
3466 332
3467 333
3468 334
3469 335
3470 336
3471 337
3472 338
3473 339
3474 340
3475 341
3476 342
3477 343
3478 344
3479 345
3480 346
3481 347
3482 348
3483 349
3484 350
3485 351
3486 352
3487 353
3488 354
3489 355
3490 356
3491 357
3492 358
3493 359
3494 360
3495 361
3496 362
3497 363
3498 364
3499 365
3500 366
3501 367
3502 368
3503 369
3504 370
3505 371
3506 372
3507 373
3508 374
3509 375
3510 376
3511 377
3512 378
3513 379
3514 380
3515 381
3516 382
3517 383
3518 384
3519 385
3520 386
3521 387
3522 388
3523 389
3524 390
3525 391
3526 392
3527 393
3528 394
3529 395
3530 396
3531 397
3532 398
3533 399
3534 400
3535 401
3536 402
3537 403
3538 404
3539 405
3540 406
3541 407
3542 408
3543 409
3544 410
3545 411
3546 412
3547 413
3548 414
3549 415
3550 416
3551 417
3552 418
3553 419
3554 420
3555 421
3556 422
3557 423
3558 424
3559 425
3560 426
3561 427
3562 428
3563 429
3564 430
3565 431
3566 432
3567 433
3568 434
3569 435
3570 436
3571 437
3572 438
3573 439
3574 440
3575 441
3576 442
3577 443
3578 444
3579 445
3580 446
3581 447
3582 448
3583 449
3584 450
3585 451
3586 452
3587 453
3588 454
3589 455
3590 456
3591 457
3592 458
3593 459
3594 460
3595 461
3596 462
3597 463
3598 464
3599 465
3600 466
3601 467
3602 468
3603 469
3604 470
3605 471
3606 472
3607 473
3608 474
3609 475
3610 476
3611 477
3612 478
3613 479
3614 480
3615 481
3616 482
3617 483
3618 484
3619 485
3620 486
3621 487
3622 488
3623 489
3624 490
3625 491
3626 492
3627 493
3628 494
3629 495
3630 496
3631 497
3632 498
3633 499
3634 500
3635 501
3636 502
3637 503
3638 504
3639 505
3640 506
3641 507
3642 508
3643 509
3644 510
3645 511
3646 512
3647 513
3648 514
3649 515
3650 516
3651 517
3652 518
3653 519
3654 520
3655 521
3656 522
3657 523
3658 524
3659 525
3660 526
3661 527
3662 528
3663 529
3664 530
3665 531
3666 532
3667 533
3668 534
3669 535
3670 536
3671 537
3672 538
3673 539
3674 540
3675 541
3676 542
3677 543
3678 544
3679 545
3680 546
3681 547
3682 548
3683 549
3684 550
3685 551
3686 552
3687 553
3688 554
3689 555
3690 556
3691 557
3692 558
3693 559
3694 560
3695 561
3696 562
3697 563
3698 564
3699 565
3700 566
3701 567
3702 568
3703 569
3704 570
3705 571
3706 572
3707 573
3708 574
3709 575
3710 576
3711 577
3712 578
3713 579
3714 580
3715 581
3716 582
3717 583
3718 584
3719 585
3720 586
3721 587
3722 588
3723 589
3724 590
3725 591
3726 592
3727 593
3728 594
3729 595
3730 596
3731 597
3732 598
3733 599
3734 600
3735 601
3736 602
3737 603
3738 604
3739 605
3740 606
3741 607
3742 608
3743 609
3744 610
3745 611
3746 612
3747 613
3748 614
3749 615
3750 616
3751 617
3752 618
3753 619
3754 620
3755 621
3756 622
3757 623
3758 624
3759 625
3760 626
3761 627
3762 628
3763 629
3764 630
3765 631
3766 632
3767 633
3768 634
3769 635
3770 636
3771 637
3772 638
3773 639
3774 640
3775 641
3776 642
3777 643
3778 644
3779 645
3780 646
3781 647
3782 648
3783 649
3784 650
3785 651
3786 652
3787 653
3788 654
3789 655
3790 656
3791 657
3792 658
3793 659
3794 660
3795 661
3796 662
3797 663
3798 664
3799 665
3800 666
3801 667
3802 668
3803 669
3804 670
3805 671
3806 672
3807 673
3808 674
3809 675
3810 676
3811 677
3812 678
3813 679
3814 680
3815 681
3816 682
3817 683
3818 684
3819 685
3820 686
3821 687
3822 688
3823 689
3824 690
3825 691
3826 692
3827 693
3828 694
3829 695
3830 696
3831 697
3832 698
3833 699
3834 700
3835 701
3836 702
3837 703
3838 704
3839 705
3840 706
3841 707
3842 708
3843 709
3844 710
3845 711
3846 712
3847 713
3848 714
3849 715
3850 716
3851 717
3852 718
3853 719
3854 720
3855 721
3856 722
3857 723
3858 724
3859 725
3860 726
3861 727
3862 728
3863 729
3864 730
3865 731
3866 732
3867 733
3868 734
3869 735
3870 736
3871 737
3872 738
3873 739
3874 740
3875 741
3876 742
3877 743
3878 744
3879 745
3880 746
3881 747
3882 748
3883 749
3884 750
3885 751
3886 752
3887 753
3888 754
3889 755
3890 756
3891 757
3892 758
3893 759
3894 760
3895 761
3896 762
3897 763
3898 764
3899 765
3900 766
3901 767
3902 768
3903 769
3904 770
3905 771
3906 772
3907 773
3908 774
3909 775
3910 776
3911 777
3912 778
3913 779
3914 780
3915 781
3916 782
3917 783
3918 784
3919 785
3920 786
3921 787
3922 788
3923 789
3924 790
3925 791
3926 792
3927 793
3928 794
3929 795
3930 796
3931 797
3932 798
3933 799
3934 800
3935 801
3936 802
3937 803
3938 804
3939 805
3940 806
3941 807
3942 808
3943 809
3944 810
3945 811
3946 812
3947 813
3948 814
3949 815
3950 816
3951 817
3952 818
3953 819
3954 820
3955 821
3956 822
3957 823
3958 824
3959 825
3960 826
3961 827
3962 828
3963 829
3964 830
3965 831
3966 832
3967 833
3968 834
3969 835
3970 836
3971 837
3972 838
3973 839
3974 840
3975 841
3976 842
3977 843
3978 844
3979 845
3980 846
3981 847
3982 848
3983 849
3984 850
3985 851
3986 852
3987 853
3988 854
3989 855
3990 856
3991 857
3992 858
3993 859
3994 860
3995 861
3996 862
3997 863
3998 864
3999 865
4000 866

```

```

3997 1  /*
3998 2  * static int DemuxAuxChildren()
3999 3  * This function handles IPC communications between the following
4000 4  * processes: RFP: Restore Engine Process, AP: auxproc, XC: xcpiogen,
4001 5  * and RC: remote command. The RC sends error and warning messages
4002 6  * up AP. When the RP is finished, RC's exit status is finished.
4003 7  * The remote exit status messages are logged and forwarded on as
4004 8  * progress messages to RP. At the same time XC will send progress
4005 9  * information to AP, and AP will forward them to RFP. XC does its
4006 10 own logging.
4007 11
4008 12 Args:
4009 13 int *progress_fd -- (in) this file descriptor is from AP to RFP.
4010 14 int remote_fd -- (in) this file descriptor is from RC to AP.
4011 15 char *remote_program -- (in) this file descriptor is from RC to AP.
4012 16 int xcpiogen_fd -- (in) this file descriptor is from XC to AP.
4013 17 int xcpiogen_pid -- (in) the pid for XC.
4014 18 int *remote_exit -- (out) the remote exit interpreted.
4015 19
4016 20 RFP: Restore Engine Process, AP: auxproc, XC: xcpiogen,
4017 21 RC: remote command.
4018 22
4019 23 * Returns: ??
4020 24
4021 25 */
4022 26
4023 27 static int
4024 28 DemuxAuxChildren(int progress_fd,
4025 29                  int remote_fd,
4026 30                  char *remote_program,
4027 31                  int xcpiogen_fd,
4028 32                  int xcpiogen_pid,
4029 33                  int *remote_exit)
4030 34 {
4031 35     int remote_exitinfo;
4032 36     int remote_exitinfo_exited = FALSE;
4033 37     struct pollfd monitor_fds[2];
4034 38     int poll_ret;
4035 39     boolean_tv remote_first_call = TRUE;
4036 40     int logging_channel = 0; /* set to global */
4037 41     int stat_rc = stat_rc; /* set to global */
4038 42     boolean_tv rc_logen_error_occurred;
4039 43     int number_fds;
4040 44     /* The below args are needed by and maintained by
4041 45      * parse_remote_status_info(). No need to initialize
4042 46      * or test them, they simply maintain state information
4043 47      * for multiple invocations of parse_remote_status_info().
4044 48      */
4045 49     enum input_status remote_state_ptr;
4046 50     enum input_status remote_next_state_ptr;
4047 51     boolean_tv skip_whitespace;
4048 52     int flags;
4049 53     int flags;
4050 54     number_fds = 2;
4051 55     monitor_fds[0].fd = remote_fd;
4052 56     monitor_fds[1].fd = events = POLLIN;
4053 57     monitor_fds[0].events = 0;
4054 58     3968 1
4055 59
4056 60
4057 61
4058 62
4059 63
4060 64
4061 65
4062 66
4063 67
4064 68
4065 69
4066 70
4067 71
4068 72
4069 73
4070 74
4071 75
4072 76
4073 77
4074 78
4075 79
4076 80
4077 81
4078 82
4079 83
4080 84
4081 85
4082 86
4083 87
4084 88
4085 89
4086 90
4087 91
4088 92
4089 93
4090 94
4091 95
4092 96
4093 97
4094 98
4095 99
4096 100
4097 101
4098 102
4099 103
4100 104
4101 105
4102 106
4103 107
4104 108
4105 109
4106 110
4107 111
4108 112
4109 113
4110 114
4111 115
4112 116
4113 117
4114 118
4115 119
4116 120
4117 121
4118 122
4119 123
4120 124
4121 125
4122 126
4123 127
4124 128
4125 129
4126 130
4127 131
4128 132
4129 133
4130 134
4131 135
4132 136
4133 137
4134 138
4135 139
4136 140
4137 141
4138 142
4139 143
4140 144
4141 145
4142 146
4143 147
4144 148
4145 149
4146 150
4147 151
4148 152
4149 153
4150 154
4151 155
4152 156
4153 157
4154 158
4155 159
4156 160
4157 161
4158 162
4159 163
4160 164
4161 165
4162 166
4163 167
4164 168
4165 169
4166 170
4167 171
4168 172
4169 173
4170 174
4171 175
4172 176
4173 177
4174 178
4175 179
4176 180
4177 181
4178 182
4179 183
4180 184
4181 185
4182 186
4183 187
4184 188
4185 189
4186 190
4187 191
4188 192
4189 193
4190 194
4191 195
4192 196
4193 197
4194 198
4195 199
4196 200
4197 201
4198 202
4199 203
4200 204
4201 205
4202 206
4203 207
4204 208
4205 209
4206 210
4207 211
4208 212
4209 213
4210 214
4211 215
4212 216
4213 217
4214 218
4215 219
4216 220
4217 221
4218 222
4219 223
4220 224
4221 225
4222 226
4223 227
4224 228
4225 229
4226 230
4227 231
4228 232
4229 233
4230 234
4231 235
4232 236
4233 237
4234 238
4235 239
4236 240
4237 241
4238 242
4239 243
4240 244
4241 245
4242 246
4243 247
4244 248
4245 249
4246 250
4247 251
4248 252
4249 253
4250 254
4251 255
4252 256
4253 257
4254 258
4255 259
4256 260
4257 261
4258 262
4259 263
4260 264
4261 265
4262 266
4263 267
4264 268
4265 269
4266 270
4267 271
4268 272
4269 273
4270 274
4271 275
4272 276
4273 277
4274 278
4275 279
4276 280
4277 281
4278 282
4279 283
4280 284
4281 285
4282 286
4283 287
4284 288
4285 289
4286 290
4287 291
4288 292
4289 293
4290 294
4291 295
4292 296
4293 297
4294 298
4295 299
4296 300
4297 301
4298 302
4299 303
4300 304
4301 305
4302 306
4303 307
4304 308
4305 309
4306 310
4307 311
4308 312
4309 313
4310 314
4311 315
4312 316
4313 317
4314 318
4315 319
4316 320
4317 321
4318 322
4319 323
4320 324
4321 325
4322 326
4323 327
4324 328
4325 329
4326 330
4327 331
4328 332
4329 333
4330 334
4331 335
4332 336
4333 337
4334 338
4335 339
4336 340
4337 341
4338 342
4339 343
4340 344
4341 345
4342 346
4343 347
4344 348
4345 349
4346 350
4347 351
4348 352
4349 353
4350 354
4351 355
4352 356
4353 357
4354 358
4355 359
4356 360
4357 361
4358 362
4359 363
4360 364
4361 365
4362 366
4363 367
4364 368
4365 369
4366 370
4367 371
4368 372
4369 373
4370 374
4371 375
4372 376
4373 377
4374 378
4375 379
4376 380
4377 381
4378 382
4379 383
4380 384
4381 385
4382 386
4383 387
4384 388
4385 389
4386 390
4387 391
4388 392
4389 393
4390 394
4391 395
4392 396
4393 397
4394 398
4395 399
4396 400
4397 401
4398 402
4399 403
4400 404
4401 405
4402 406
4403 407
4404 408
4405 409
4406 410
4407 411
4408 412
4409 413
4410 414
4411 415
4412 416
4
```


D12

```

RSI_Init 5 (RSI_Init.c)
RSI_Init_Validate.....3 (RSI_Init.c)
init_plugin 9 (RSI_Init.c)
validate_plugin.....13 (RSI_Init.c)

```

```

2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61

/*.....*/
** File Name:  RSInitfin.c
**
** Copyright (c) 1998,1999 by EMC Corporation.
**
** Purpose:   This module contains the Restore Service Library
              functions to
              initialize and terminate the restore operation.
**
** Table of Contents:
**
**      -----
**      RSInit, Initialize
**      RSTSL, Finish
**
** Internal Functions:
**
**
** Compile-time Options:
** This section must list any compile time definitions
** which will affect this header.
**
**.....*/

/* The following provides an RCS id in the binary that can be located
** with the whac(1) utility.  The intent is to keep this short.
**/

#ifndef LINE
static char RCS_id[] = "RSInitfin.c"
#define RCS_id RCS_id
#endif

/*
** Procedure test switches.
** Standard defines required to turn on OS features go here.
** The following is required for code that uses POSIX API's.
** Remove for non-POSIX, non-portable code.
**/

#define _POSIX_SOURCE 1

/*
** * System headers.
**/
#include <sys/param.h>
#include <fcntl.h>
#include <fcntl.h>

/*
** * Epoch headers.
**/
#include <sys/cb_port.h>
#include <cb/rb_log.h>

```

```

54  /*
55   * Local headers
56   */
57  #include "asInterna.h"
58  #include "asIatrap.h"
59
60  /*
61   * Definitions of the plugin functions in the pifuncarray
62   * of the plugdata structure.
63   * These must be in the same order and position
64   * as the pifuncindex values defined in NstIplugin.h.
65   */
66
67  char *pifuncnames[pifuncindex+1] = {
68      "asIPluginInit",
69      "asIPluginInitialize",
70      "asIPluginGetLevelObject",
71      "asIPluginGetNextLevelObject",
72      "asIPluginClearNextLevelContext",
73      "asIPluginSubmit",
74      "asIPluginGetLevelTemplate",
75      "asIPluginDoasIatrapExit",
76      "asIPluginMarkObject",
77      "asIPluginIsObjectMarked",
78      "asIPluginGetObjBackPrime",
79      "asIPluginGetObjBackPrimeName",
80      "asIPluginGetObjBackPrimeTime",
81      "asIPluginGetPrevBackup",
82      "asIPluginGetNextBackup",
83      "asIPluginGetFirstBackup",
84      "asIPluginGetMostRecentBackup",
85      "asIPluginGetIatrapBackup",
86      "asIPluginGetIatrapBackupPortTime",
87      "asIPluginGetIatrapBackupPortTime",
88      "asIPluginStartIatrap",
89      "asIPluginFindObjects",
90      "asIPluginGetFindResults",
91      "asIPluginGetNecessaryMedia"
92  };
93
94  }

```

```

1277  /* ===== */
1278  /* RECOVER_Initialize: */
1279  /* ===== */
1280  /* This function takes care of all the initialization for a restore
1281  session. This must be called prior to any of the other functions
1282  in the Restore API. */
1283  /* Parameters: */
1284  /*     username (I) - The name of the user.
1285  *=====*/
1286  /*
1287  *=====*/
1288  /*
1289  *=====*/
1290  /*
1291  *=====*/
1292  /*
1293  *=====*/
1294  /*
1295  *=====*/
1296  /*
1297  *=====*/
1298  /*
1299  *=====*/
1300  /*
1301  *=====*/
1302  /*
1303  *=====*/
1304  /*
1305  *=====*/
1306  /*
1307  *=====*/
1308  /*
1309  *=====*/
1310  /*
1311  *=====*/
1312  /*
1313  *=====*/
1314  /*
1315  *=====*/
1316  /*
1317  *=====*/
1318  /*
1319  *=====*/
1320  /*
1321  *=====*/
1322  /*
1323  *=====*/
1324  /*
1325  *=====*/
1326  /*
1327  *=====*/
1328  /*
1329  *=====*/
1330  /*
1331  *=====*/
1332  /*
1333  *=====*/
1334  /*
1335  *=====*/
1336  /*
1337  *=====*/
1338  /*
1339  *=====*/
1340  /*
1341  *=====*/
1342  /*
1343  *=====*/
1344  /*
1345  *=====*/
1346  /*
1347  *=====*/
1348  /*
1349  *=====*/
1350  /*
1351  *=====*/
1352  /*
1353  *=====*/
1354  /*
1355  *=====*/
1356  /*
1357  *=====*/
1358  /*
1359  *=====*/
1360  /*
1361  *=====*/
1362  /*
1363  *=====*/
1364  /*
1365  *=====*/
1366  /*
1367  *=====*/
1368  /*
1369  *=====*/
1370  /*
1371  *=====*/
1372  /*
1373  *=====*/
1374  /*
1375  *=====*/
1376  /*
1377  *=====*/
1378  /*
1379  *=====*/
1380  /*
1381  *=====*/
1382  /*
1383  *=====*/
1384  /*
1385  *=====*/
1386  /*
1387  *=====*/
1388  /*
1389  *=====*/
1390  /*
1391  *=====*/
1392  /*
1393  *=====*/
1394  /*
1395  *=====*/
1396  /*
1397  *=====*/
1398  /*
1399  *=====*/
1400  /*
1401  *=====*/
1402  /*
1403  *=====*/
1404  /*
1405  *=====*/
1406  /*
1407  *=====*/
1408  /*
1409  *=====*/
1410  /*
1411  *=====*/
1412  /*
1413  *=====*/
1414  /*
1415  *=====*/
1416  /*
1417  *=====*/
1418  /*
1419  *=====*/
1420  /*
1421  *=====*/
1422  /*
1423  *=====*/
1424  /*
1425  *=====*/
1426  /*
1427  *=====*/
1428  /*
1429  *=====*/
1430  /*
1431  *=====*/
1432  /*
1433  *=====*/
1434  /*
1435  *=====*/
1436  /*
1437  *=====*/
1438  /*
1439  *=====*/
1440  /*
1441  *=====*/
1442  /*
1443  *=====*/
1444  /*
1445  *=====*/
1446  /*
1447  *=====*/
1448  /*
1449  *=====*/
1450  /*
1451  *=====*/
1452  /*
1453  *=====*/
1454  /*
1455  *=====*/
1456  /*
1457  *=====*/
1458  /*
1459  *=====*/
1460  /*
1461  *=====*/
1462  /*
1463  *=====*/
1464  /*
1465  *=====*/
1466  /*
1467  *=====*/
1468  /*
1469  *=====*/
1470  /*
1471  *=====*/
1472  /*
1473  *=====*/
1474  /*
1475  *=====*/
1476  /*
1477  *=====*/
1478  /*
1479  *=====*/
1480  /*
1481  *=====*/
1482  /*
1483  *=====*/
1484  /*
1485  *=====*/
1486  /*
1487  *=====*/
1488  /*
1489  *=====*/
1490  /*
1491  *=====*/
1492  /*
1493  *=====*/
1494  /*
1495  *=====*/
1496  /*
1497  *=====*/
1498  /*
1499  *=====*/
1500  /*
1501  *=====*/
1502  /*
1503  *=====*/
1504  /*
1505  *=====*/
1506  /*
1507  *=====*/
1508  /*
1509  *=====*/
1510  /*
1511  *=====*/
1512  /*
1513  *=====*/
1514  /*
1515  *=====*/
1516  /*
1517  *=====*/
1518  /*
1519  *=====*/
1520  /*
1521  *=====*/
1522  /*
1523  *=====*/
1524  /*
1525  *=====*/
1526  /*
1527  *=====*/
1528  /*
1529  *=====*/
1530  /*
1531  *=====*/
1532  /*
1533  *=====*/
1534  /*
1535  *=====*/
1536  /*
1537  *=====*/
1538  /*
1539  *=====*/
1540  /*
1541  *=====*/
1542  /*
1543  *=====*/
1544  /*
1545  *=====*/
1546  /*
1547  *=====*/
1548  /*
1549  *=====*/
1550  /*
1551  *=====*/
1552  /*
1553  *=====*/
1554  /*
1555  *=====*/
1556  /*
1557  *=====*/
1558  /*
1559  *=====*/
1560  /*
1561  *=====*/
1562  /*
1563  *=====*/
1564  /*
1565  *=====*/
1566  /*
1567  *=====*/
1568  /*
1569  *=====*/
1570  /*
1571  *=====*/
1572  /*
1573  *=====*/
1574  /*
1575  *=====*/
1576  /*
1577  *=====*/
1578  /*
1579  *=====*/
1580  /*
1581  *=====*/
1582  /*
1583  *=====*/
1584  /*
1585  *=====*/
1586  /*
1587  *=====*/
1588  /*
1589  *=====*/
1590  /*
1591  *=====*/
1592  /*
1593  *=====*/
1594  /*
1595  *=====*/
1596  /*
1597  *=====*/
1598  /*
1599  *=====*/
1600  /*
1601  *=====*/
1602  /*
1603  *=====*/
1604  /*
1605  *=====*/
1606  /*
1607  *=====*/
1608  /*
1609  *=====*/
1610  /*
1611  *=====*/
1612  /*
1613  *=====*/
1614  /*
1615  *=====*/
1616  /*
1617  *=====*/
1618  /*
1619  *=====*/
1620  /*
1621  *=====*/
1622  /*
1623  *=====*/
1624  /*
1625  *=====*/
1626  /*
1627  *=====*/
1628  /*
1629  *=====*/
1630  /*
1631  *=====*/
1632  /*
1633  *=====*/
1634  /*
1635  *=====*/
1636  /*
1637  *=====*/
1638  /*
1639  *=====*/
1640  /*
1641  *=====*/
1642  /*
1643  *=====*/
1644  /*
1645  *=====*/
1646  /*
1647  *=====*/
1648  /*
1649  *=====*/
1650  /*
1651  *=====*/
1652  /*
1653  *=====*/
1654  /*
1655  *=====*/
1656  /*
1657  *=====*/
1658  /*
1659  *=====*/
1660  /*
1661  *=====*/
1662  /*
1663  *=====*/
1664  /*
1665  *=====*/
1666  /*
1667  *=====*/
1668  /*
1669  *=====*/
1670  /*
1671  *=====*/
1672  /*
1673  *=====*/
1674  /*
1675  *=====*/
1676  /*
1677  *=====*/
1678  /*
1679  *=====*/
1680  /*
1681  *=====*/
1682  /*
1683  *=====*/
1684  /*
1685  *=====*/
1686  /*
1687  *=====*/
1688  /*
1689  *=====*/
1690  /*
1691  *=====*/
1692  /*
1693  *=====*/
1694  /*
1695  *=====*/
1696  /*
1697  *=====*/
1698  /*
1699  *=====*/
1700  /*
1701  *=====*/
1702  /*
1703  *=====*/
1704  /*
1705  *=====*/
1706  /*
1707  *=====*/
1708  /*
1709  *=====*/
1710  /*
1711  *=====*/
1712  /*
1713  *=====*/
1714  /*
1715  *=====*/
1716  /*
1717  *=====*/
1718  /*
1719  *=====*/
1720  /*
1721  *=====*/
1722  /*
1723  *=====*/
1724  /*
1725  *=====*/
1726  /*
1727  *=====*/
1728  /*
1729  *=====*/
1730  /*
1731  *=====*/
1732  /*
1733  *=====*/
1734  /*
1735  *=====*/
1736  /*
1737  *=====*/
1738  /*
1739  *=====*/
1740  /*
1741  *=====*/
1742  /*
1743  *=====*/
1744  /*
1745  *=====*/
1746  /*
1747  *=====*/
1748  /*
1749  *=====*/
1750  /*
1751  *=====*/
1752  /*
1753  *=====*/
1754  /*
1755  *=====*/
1756  /*
1757  *=====*/
1758  /*
1759  *=====*/
1760  /*
1761  *=====*/
1762  /*
1763  *=====*/
1764  /*
1765  *=====*/
1766  /*
1767  *=====*/
1768  /*
1769  *=====*/
1770  /*
1771  *=====*/
1772  /*
1773  *=====*/
1774  /*
1775  *=====*/
1776  /*
1777  *=====*/
1778  /*
1779  *=====*/
1780  /*
1781  *=====*/
1782  /*

```

```

191. 1      /*
192. 1      setup_proc(top) ;
193. 1
194. 1      /* The following call will:
195. 1      *   - Initialize the server database.
196. 1      *   - Insert any information we can at this point.
197. 1      */
198. 1      if (status == setup_proc)
199. 1      {
200. 1          return(status);
201. 1      }
202. 1
203. 1      /* Do plugins setup: Find and initialize all valid resource plugin
204. 1      libraries.
205. 1
206. 1      status = init_plugins(top) ;
207. 1
208. 1      return(status) ;
209. 1      /* End of RSVSL_Initialize() */
210. 1
211. 1 }

```

Mon Oct 13 16:39:31 2008

```

214 /***** RSTSL_Finish *****/
215
216 * Function Description:
217
218 * This function terminates a restore session,
219   but not while a restore is in
220   progress.
221   It will be rejected if a restore is currently being executed.
222   This routine will clean up any local memory used in the session.
223
224 * Parameters:
225 * none
226
227 */
228
229 restore_cleanup(void)
230 {
231     int rc;
232
233     restore_cleanup_err = E_SUCCESS;
234
235     if (NULL != rc)
236     {
237         return( E_SUCCESS );
238     }
239     /* Remove all the structures that are currently in use. */
240     /* Call the cleanup() which kills the aux proc(s), unlocks the work
241        * item, then calls rclog_end() to enter the last logs and to close
242        * the log file.
243
244     */
245     rc_cleanup(rc);
246
247     /* Deallocate the memory of restore context and the related
248        * structures.
249
250     */
251     if (NULL != rc)
252     {
253         /* Free the multilevel structures */
254         multilevel_cleanup(rc);
255     }
256     /* Free the mark bit map space
257
258     */
259     for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
260     {
261         if (NULL != rc->rc_marks[mc_n])
262         {
263             free(rc->rc_marks[mc_n]);
264         }
265         rc->rc_marks[mc_n] = NULL;
266     }
267     if (NULL != rc)
268     {
269         /* Free the backup app = 0;
270            while (rc->currentFidT = rc->pliset)
271            {
272                rc->rc_backup_app++;
273                rc->appData = rc->currentFidT->appData;
274                /* Allow plugin to clean up and close .so: */
275                if ( E_SUCCESS != err =
276                    rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
277                {
278                    /* log error, continue */
279                    rc->user_error[ err ];
280                    RSTSL_Finish_failed_for_restore_plugin(
281                        (struct pluginData *) (
282                            rc->currentFidT->idData )->name );
283                }
284                rc->pliset = rc->pliset->next;
285            }
286            rc->currentFidT = NULL;
287        }
288    }
289
290    if (NULL != rc)
291    {
292        /* Free the volume list structures.
293
294        */
295        if (NULL != rc->vol_list)
296        {
297            free(rc->vol_list);
298        }
299        /* Free the plugin related data
300
301        */
302        if (NULL != rc->plugin_data)
303        {
304            (void) bvol_volidlist_destructor(
305                rc->vol_list, BVOL_DESTROY_ASYN);
306        }
307    }
308
309    /* Free the backup app = 0;
310       while (rc->currentFidT = rc->pliset)
311       {
312           rc->rc_backup_app++;
313           rc->appData = rc->currentFidT->appData;
314           /* Allow plugin to clean up and close .so: */
315           if ( E_SUCCESS != err =
316               rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
317           {
318               /* log error, continue */
319               rc->user_error[ err ];
320               RSTSL_Finish_failed_for_restore_plugin(
321                   (struct pluginData *) (
322                       rc->currentFidT->idData )->name );
323           }
324           rc->pliset = rc->pliset->next;
325       }
326       rc->currentFidT = NULL;
327   }
328
329   if (NULL != rc)
330   {
331       /* Free the multilevel structures */
332       multilevel_cleanup(rc);
333   }
334   /* Free the mark bit map space
335
336   */
337   for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
338   {
339       if (NULL != rc->rc_marks[mc_n])
340       {
341           free(rc->rc_marks[mc_n]);
342       }
343       rc->rc_marks[mc_n] = NULL;
344   }
345   if (NULL != rc)
346   {
347       /* Free the backup app = 0;
348          while (rc->currentFidT = rc->pliset)
349          {
350              rc->rc_backup_app++;
351              rc->appData = rc->currentFidT->appData;
352              /* Allow plugin to clean up and close .so: */
353              if ( E_SUCCESS != err =
354                  rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
355              {
356                  /* log error, continue */
357                  rc->user_error[ err ];
358                  RSTSL_Finish_failed_for_restore_plugin(
359                      (struct pluginData *) (
360                          rc->currentFidT->idData )->name );
361              }
362              rc->pliset = rc->pliset->next;
363          }
364          rc->currentFidT = NULL;
365      }
366  }
367
368  if (NULL != rc)
369  {
370      /* Free the volume list structures.
371
372      */
373      if (NULL != rc->vol_list)
374      {
375          free(rc->vol_list);
376      }
377      /* Free the plugin related data
378
379      */
380      if (NULL != rc->plugin_data)
381      {
382          (void) bvol_volidlist_destructor(
383              rc->vol_list, BVOL_DESTROY_ASYN);
384      }
385  }
386
387  /* Free the backup app = 0;
388     while (rc->currentFidT = rc->pliset)
389     {
390         rc->rc_backup_app++;
391         rc->appData = rc->currentFidT->appData;
392         /* Allow plugin to clean up and close .so: */
393         if ( E_SUCCESS != err =
394             rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
395         {
396             /* log error, continue */
397             rc->user_error[ err ];
398             RSTSL_Finish_failed_for_restore_plugin(
399                 (struct pluginData *) (
400                     rc->currentFidT->idData )->name );
401         }
402         rc->pliset = rc->pliset->next;
403     }
404     rc->currentFidT = NULL;
405 }
406
407 if (NULL != rc)
408 {
409     /* Free the multilevel structures */
410     multilevel_cleanup(rc);
411 }
412 /* Free the mark bit map space
413
414 */
415 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
416 {
417     if (NULL != rc->rc_marks[mc_n])
418     {
419         free(rc->rc_marks[mc_n]);
420     }
421     rc->rc_marks[mc_n] = NULL;
422 }
423 if (NULL != rc)
424 {
425     /* Free the backup app = 0;
426        while (rc->currentFidT = rc->pliset)
427        {
428            rc->rc_backup_app++;
429            rc->appData = rc->currentFidT->appData;
430            /* Allow plugin to clean up and close .so: */
431            if ( E_SUCCESS != err =
432                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
433            {
434                /* log error, continue */
435                rc->user_error[ err ];
436                RSTSL_Finish_failed_for_restore_plugin(
437                    (struct pluginData *) (
438                        rc->currentFidT->idData )->name );
439            }
440            rc->pliset = rc->pliset->next;
441        }
442        rc->currentFidT = NULL;
443    }
444 }
445
446 if (NULL != rc)
447 {
448     /* Free the volume list structures.
449
450     */
451     if (NULL != rc->vol_list)
452     {
453         free(rc->vol_list);
454     }
455     /* Free the plugin related data
456
457     */
458     if (NULL != rc->plugin_data)
459     {
460         (void) bvol_volidlist_destructor(
461             rc->vol_list, BVOL_DESTROY_ASYN);
462     }
463 }
464
465 /* Free the backup app = 0;
466    while (rc->currentFidT = rc->pliset)
467    {
468        rc->rc_backup_app++;
469        rc->appData = rc->currentFidT->appData;
470        /* Allow plugin to clean up and close .so: */
471        if ( E_SUCCESS != err =
472            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
473        {
474            /* log error, continue */
475            rc->user_error[ err ];
476            RSTSL_Finish_failed_for_restore_plugin(
477                (struct pluginData *) (
478                    rc->currentFidT->idData )->name );
479        }
480        rc->pliset = rc->pliset->next;
481    }
482    rc->currentFidT = NULL;
483 }
484
485 if (NULL != rc)
486 {
487     /* Free the multilevel structures */
488     multilevel_cleanup(rc);
489 }
490 /* Free the mark bit map space
491
492 */
493 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
494 {
495     if (NULL != rc->rc_marks[mc_n])
496     {
497         free(rc->rc_marks[mc_n]);
498     }
499     rc->rc_marks[mc_n] = NULL;
500 }
501 if (NULL != rc)
502 {
503     /* Free the backup app = 0;
504        while (rc->currentFidT = rc->pliset)
505        {
506            rc->rc_backup_app++;
507            rc->appData = rc->currentFidT->appData;
508            /* Allow plugin to clean up and close .so: */
509            if ( E_SUCCESS != err =
510                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
511            {
512                /* log error, continue */
513                rc->user_error[ err ];
514                RSTSL_Finish_failed_for_restore_plugin(
515                    (struct pluginData *) (
516                        rc->currentFidT->idData )->name );
517            }
518            rc->pliset = rc->pliset->next;
519        }
520        rc->currentFidT = NULL;
521    }
522 }
523
524 if (NULL != rc)
525 {
526     /* Free the volume list structures.
527
528     */
529     if (NULL != rc->vol_list)
530     {
531         free(rc->vol_list);
532     }
533     /* Free the plugin related data
534
535     */
536     if (NULL != rc->plugin_data)
537     {
538         (void) bvol_volidlist_destructor(
539             rc->vol_list, BVOL_DESTROY_ASYN);
540     }
541 }
542
543 /* Free the backup app = 0;
544    while (rc->currentFidT = rc->pliset)
545    {
546        rc->rc_backup_app++;
547        rc->appData = rc->currentFidT->appData;
548        /* Allow plugin to clean up and close .so: */
549        if ( E_SUCCESS != err =
550            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
551        {
552            /* log error, continue */
553            rc->user_error[ err ];
554            RSTSL_Finish_failed_for_restore_plugin(
555                (struct pluginData *) (
556                    rc->currentFidT->idData )->name );
557        }
558        rc->pliset = rc->pliset->next;
559    }
560    rc->currentFidT = NULL;
561 }
562
563 if (NULL != rc)
564 {
565     /* Free the multilevel structures */
566     multilevel_cleanup(rc);
567 }
568 /* Free the mark bit map space
569
570 */
571 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
572 {
573     if (NULL != rc->rc_marks[mc_n])
574     {
575         free(rc->rc_marks[mc_n]);
576     }
577     rc->rc_marks[mc_n] = NULL;
578 }
579 if (NULL != rc)
580 {
581     /* Free the backup app = 0;
582        while (rc->currentFidT = rc->pliset)
583        {
584            rc->rc_backup_app++;
585            rc->appData = rc->currentFidT->appData;
586            /* Allow plugin to clean up and close .so: */
587            if ( E_SUCCESS != err =
588                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
589            {
590                /* log error, continue */
591                rc->user_error[ err ];
592                RSTSL_Finish_failed_for_restore_plugin(
593                    (struct pluginData *) (
594                        rc->currentFidT->idData )->name );
595            }
596            rc->pliset = rc->pliset->next;
597        }
598        rc->currentFidT = NULL;
599    }
600 }
601
602 if (NULL != rc)
603 {
604     /* Free the volume list structures.
605
606     */
607     if (NULL != rc->vol_list)
608     {
609         free(rc->vol_list);
610     }
611     /* Free the plugin related data
612
613     */
614     if (NULL != rc->plugin_data)
615     {
616         (void) bvol_volidlist_destructor(
617             rc->vol_list, BVOL_DESTROY_ASYN);
618     }
619 }
620
621 /* Free the backup app = 0;
622    while (rc->currentFidT = rc->pliset)
623    {
624        rc->rc_backup_app++;
625        rc->appData = rc->currentFidT->appData;
626        /* Allow plugin to clean up and close .so: */
627        if ( E_SUCCESS != err =
628            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
629        {
630            /* log error, continue */
631            rc->user_error[ err ];
632            RSTSL_Finish_failed_for_restore_plugin(
633                (struct pluginData *) (
634                    rc->currentFidT->idData )->name );
635        }
636        rc->pliset = rc->pliset->next;
637    }
638    rc->currentFidT = NULL;
639 }
640
641 if (NULL != rc)
642 {
643     /* Free the multilevel structures */
644     multilevel_cleanup(rc);
645 }
646 /* Free the mark bit map space
647
648 */
649 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
650 {
651     if (NULL != rc->rc_marks[mc_n])
652     {
653         free(rc->rc_marks[mc_n]);
654     }
655     rc->rc_marks[mc_n] = NULL;
656 }
657 if (NULL != rc)
658 {
659     /* Free the backup app = 0;
660        while (rc->currentFidT = rc->pliset)
661        {
662            rc->rc_backup_app++;
663            rc->appData = rc->currentFidT->appData;
664            /* Allow plugin to clean up and close .so: */
665            if ( E_SUCCESS != err =
666                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
667            {
668                /* log error, continue */
669                rc->user_error[ err ];
670                RSTSL_Finish_failed_for_restore_plugin(
671                    (struct pluginData *) (
672                        rc->currentFidT->idData )->name );
673            }
674            rc->pliset = rc->pliset->next;
675        }
676        rc->currentFidT = NULL;
677    }
678 }
679
680 if (NULL != rc)
681 {
682     /* Free the volume list structures.
683
684     */
685     if (NULL != rc->vol_list)
686     {
687         free(rc->vol_list);
688     }
689     /* Free the plugin related data
690
691     */
692     if (NULL != rc->plugin_data)
693     {
694         (void) bvol_volidlist_destructor(
695             rc->vol_list, BVOL_DESTROY_ASYN);
696     }
697 }
698
699 /* Free the backup app = 0;
700    while (rc->currentFidT = rc->pliset)
701    {
702        rc->rc_backup_app++;
703        rc->appData = rc->currentFidT->appData;
704        /* Allow plugin to clean up and close .so: */
705        if ( E_SUCCESS != err =
706            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
707        {
708            /* log error, continue */
709            rc->user_error[ err ];
710            RSTSL_Finish_failed_for_restore_plugin(
711                (struct pluginData *) (
712                    rc->currentFidT->idData )->name );
713        }
714        rc->pliset = rc->pliset->next;
715    }
716    rc->currentFidT = NULL;
717 }
718
719 if (NULL != rc)
720 {
721     /* Free the multilevel structures */
722     multilevel_cleanup(rc);
723 }
724 /* Free the mark bit map space
725
726 */
727 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
728 {
729     if (NULL != rc->rc_marks[mc_n])
730     {
731         free(rc->rc_marks[mc_n]);
732     }
733     rc->rc_marks[mc_n] = NULL;
734 }
735 if (NULL != rc)
736 {
737     /* Free the backup app = 0;
738        while (rc->currentFidT = rc->pliset)
739        {
740            rc->rc_backup_app++;
741            rc->appData = rc->currentFidT->appData;
742            /* Allow plugin to clean up and close .so: */
743            if ( E_SUCCESS != err =
744                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
745            {
746                /* log error, continue */
747                rc->user_error[ err ];
748                RSTSL_Finish_failed_for_restore_plugin(
749                    (struct pluginData *) (
750                        rc->currentFidT->idData )->name );
751            }
752            rc->pliset = rc->pliset->next;
753        }
754        rc->currentFidT = NULL;
755    }
756 }
757
758 if (NULL != rc)
759 {
760     /* Free the volume list structures.
761
762     */
763     if (NULL != rc->vol_list)
764     {
765         free(rc->vol_list);
766     }
767     /* Free the plugin related data
768
769     */
770     if (NULL != rc->plugin_data)
771     {
772         (void) bvol_volidlist_destructor(
773             rc->vol_list, BVOL_DESTROY_ASYN);
774     }
775 }
776
777 /* Free the backup app = 0;
778    while (rc->currentFidT = rc->pliset)
779    {
780        rc->rc_backup_app++;
781        rc->appData = rc->currentFidT->appData;
782        /* Allow plugin to clean up and close .so: */
783        if ( E_SUCCESS != err =
784            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
785        {
786            /* log error, continue */
787            rc->user_error[ err ];
788            RSTSL_Finish_failed_for_restore_plugin(
789                (struct pluginData *) (
790                    rc->currentFidT->idData )->name );
791        }
792        rc->pliset = rc->pliset->next;
793    }
794    rc->currentFidT = NULL;
795 }
796
797 if (NULL != rc)
798 {
799     /* Free the multilevel structures */
800     multilevel_cleanup(rc);
801 }
802 /* Free the mark bit map space
803
804 */
805 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
806 {
807     if (NULL != rc->rc_marks[mc_n])
808     {
809         free(rc->rc_marks[mc_n]);
810     }
811     rc->rc_marks[mc_n] = NULL;
812 }
813 if (NULL != rc)
814 {
815     /* Free the backup app = 0;
816        while (rc->currentFidT = rc->pliset)
817        {
818            rc->rc_backup_app++;
819            rc->appData = rc->currentFidT->appData;
820            /* Allow plugin to clean up and close .so: */
821            if ( E_SUCCESS != err =
822                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
823            {
824                /* log error, continue */
825                rc->user_error[ err ];
826                RSTSL_Finish_failed_for_restore_plugin(
827                    (struct pluginData *) (
828                        rc->currentFidT->idData )->name );
829            }
830            rc->pliset = rc->pliset->next;
831        }
832        rc->currentFidT = NULL;
833    }
834 }
835
836 if (NULL != rc)
837 {
838     /* Free the volume list structures.
839
840     */
841     if (NULL != rc->vol_list)
842     {
843         free(rc->vol_list);
844     }
845     /* Free the plugin related data
846
847     */
848     if (NULL != rc->plugin_data)
849     {
850         (void) bvol_volidlist_destructor(
851             rc->vol_list, BVOL_DESTROY_ASYN);
852     }
853 }
854
855 /* Free the backup app = 0;
856    while (rc->currentFidT = rc->pliset)
857    {
858        rc->rc_backup_app++;
859        rc->appData = rc->currentFidT->appData;
860        /* Allow plugin to clean up and close .so: */
861        if ( E_SUCCESS != err =
862            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
863        {
864            /* log error, continue */
865            rc->user_error[ err ];
866            RSTSL_Finish_failed_for_restore_plugin(
867                (struct pluginData *) (
868                    rc->currentFidT->idData )->name );
869        }
870        rc->pliset = rc->pliset->next;
871    }
872    rc->currentFidT = NULL;
873 }
874
875 if (NULL != rc)
876 {
877     /* Free the multilevel structures */
878     multilevel_cleanup(rc);
879 }
880 /* Free the mark bit map space
881
882 */
883 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
884 {
885     if (NULL != rc->rc_marks[mc_n])
886     {
887         free(rc->rc_marks[mc_n]);
888     }
889     rc->rc_marks[mc_n] = NULL;
890 }
891 if (NULL != rc)
892 {
893     /* Free the backup app = 0;
894        while (rc->currentFidT = rc->pliset)
895        {
896            rc->rc_backup_app++;
897            rc->appData = rc->currentFidT->appData;
898            /* Allow plugin to clean up and close .so: */
899            if ( E_SUCCESS != err =
900                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
901            {
902                /* log error, continue */
903                rc->user_error[ err ];
904                RSTSL_Finish_failed_for_restore_plugin(
905                    (struct pluginData *) (
906                        rc->currentFidT->idData )->name );
907            }
908            rc->pliset = rc->pliset->next;
909        }
910        rc->currentFidT = NULL;
911    }
912 }
913
914 if (NULL != rc)
915 {
916     /* Free the volume list structures.
917
918     */
919     if (NULL != rc->vol_list)
920     {
921         free(rc->vol_list);
922     }
923     /* Free the plugin related data
924
925     */
926     if (NULL != rc->plugin_data)
927     {
928         (void) bvol_volidlist_destructor(
929             rc->vol_list, BVOL_DESTROY_ASYN);
930     }
931 }
932
933 /* Free the backup app = 0;
934    while (rc->currentFidT = rc->pliset)
935    {
936        rc->rc_backup_app++;
937        rc->appData = rc->currentFidT->appData;
938        /* Allow plugin to clean up and close .so: */
939        if ( E_SUCCESS != err =
940            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
941        {
942            /* log error, continue */
943            rc->user_error[ err ];
944            RSTSL_Finish_failed_for_restore_plugin(
945                (struct pluginData *) (
946                    rc->currentFidT->idData )->name );
947        }
948        rc->pliset = rc->pliset->next;
949    }
950    rc->currentFidT = NULL;
951 }
952
953 if (NULL != rc)
954 {
955     /* Free the multilevel structures */
956     multilevel_cleanup(rc);
957 }
958 /* Free the mark bit map space
959
960 */
961 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
962 {
963     if (NULL != rc->rc_marks[mc_n])
964     {
965         free(rc->rc_marks[mc_n]);
966     }
967     rc->rc_marks[mc_n] = NULL;
968 }
969 if (NULL != rc)
970 {
971     /* Free the backup app = 0;
972        while (rc->currentFidT = rc->pliset)
973        {
974            rc->rc_backup_app++;
975            rc->appData = rc->currentFidT->appData;
976            /* Allow plugin to clean up and close .so: */
977            if ( E_SUCCESS != err =
978                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
979            {
980                /* log error, continue */
981                rc->user_error[ err ];
982                RSTSL_Finish_failed_for_restore_plugin(
983                    (struct pluginData *) (
984                        rc->currentFidT->idData )->name );
985            }
986            rc->pliset = rc->pliset->next;
987        }
988        rc->currentFidT = NULL;
989    }
990 }
991
992 if (NULL != rc)
993 {
994     /* Free the volume list structures.
995
996     */
997     if (NULL != rc->vol_list)
998     {
999         free(rc->vol_list);
1000     }
1001     /* Free the plugin related data
1002
1003     */
1004     if (NULL != rc->plugin_data)
1005     {
1006         (void) bvol_volidlist_destructor(
1007             rc->vol_list, BVOL_DESTROY_ASYN);
1008     }
1009 }
1010
1011 /* Free the backup app = 0;
1012    while (rc->currentFidT = rc->pliset)
1013    {
1014        rc->rc_backup_app++;
1015        rc->appData = rc->currentFidT->appData;
1016        /* Allow plugin to clean up and close .so: */
1017        if ( E_SUCCESS != err =
1018            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
1019        {
1020            /* log error, continue */
1021            rc->user_error[ err ];
1022            RSTSL_Finish_failed_for_restore_plugin(
1023                (struct pluginData *) (
1024                    rc->currentFidT->idData )->name );
1025        }
1026        rc->pliset = rc->pliset->next;
1027    }
1028    rc->currentFidT = NULL;
1029 }
1030
1031 if (NULL != rc)
1032 {
1033     /* Free the multilevel structures */
1034     multilevel_cleanup(rc);
1035 }
1036 /* Free the mark bit map space
1037
1038 */
1039 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
1040 {
1041     if (NULL != rc->rc_marks[mc_n])
1042     {
1043         free(rc->rc_marks[mc_n]);
1044     }
1045     rc->rc_marks[mc_n] = NULL;
1046 }
1047 if (NULL != rc)
1048 {
1049     /* Free the backup app = 0;
1050        while (rc->currentFidT = rc->pliset)
1051        {
1052            rc->rc_backup_app++;
1053            rc->appData = rc->currentFidT->appData;
1054            /* Allow plugin to clean up and close .so: */
1055            if ( E_SUCCESS != err =
1056                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
1057            {
1058                /* log error, continue */
1059                rc->user_error[ err ];
1060                RSTSL_Finish_failed_for_restore_plugin(
1061                    (struct pluginData *) (
1062                        rc->currentFidT->idData )->name );
1063            }
1064            rc->pliset = rc->pliset->next;
1065        }
1066        rc->currentFidT = NULL;
1067    }
1068 }
1069
1070 if (NULL != rc)
1071 {
1072     /* Free the volume list structures.
1073
1074     */
1075     if (NULL != rc->vol_list)
1076     {
1077         free(rc->vol_list);
1078     }
1079     /* Free the plugin related data
1080
1081     */
1082     if (NULL != rc->plugin_data)
1083     {
1084         (void) bvol_volidlist_destructor(
1085             rc->vol_list, BVOL_DESTROY_ASYN);
1086     }
1087 }
1088
1089 /* Free the backup app = 0;
1090    while (rc->currentFidT = rc->pliset)
1091    {
1092        rc->rc_backup_app++;
1093        rc->appData = rc->currentFidT->appData;
1094        /* Allow plugin to clean up and close .so: */
1095        if ( E_SUCCESS != err =
1096            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
1097        {
1098            /* log error, continue */
1099            rc->user_error[ err ];
1100            RSTSL_Finish_failed_for_restore_plugin(
1101                (struct pluginData *) (
1102                    rc->currentFidT->idData )->name );
1103        }
1104        rc->pliset = rc->pliset->next;
1105    }
1106    rc->currentFidT = NULL;
1107 }
1108
1109 if (NULL != rc)
1110 {
1111     /* Free the multilevel structures */
1112     multilevel_cleanup(rc);
1113 }
1114 /* Free the mark bit map space
1115
1116 */
1117 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
1118 {
1119     if (NULL != rc->rc_marks[mc_n])
1120     {
1121         free(rc->rc_marks[mc_n]);
1122     }
1123     rc->rc_marks[mc_n] = NULL;
1124 }
1125 if (NULL != rc)
1126 {
1127     /* Free the backup app = 0;
1128        while (rc->currentFidT = rc->pliset)
1129        {
1130            rc->rc_backup_app++;
1131            rc->appData = rc->currentFidT->appData;
1132            /* Allow plugin to clean up and close .so: */
1133            if ( E_SUCCESS != err =
1134                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
1135            {
1136                /* log error, continue */
1137                rc->user_error[ err ];
1138                RSTSL_Finish_failed_for_restore_plugin(
1139                    (struct pluginData *) (
1140                        rc->currentFidT->idData )->name );
1141            }
1142            rc->pliset = rc->pliset->next;
1143        }
1144        rc->currentFidT = NULL;
1145    }
1146 }
1147
1148 if (NULL != rc)
1149 {
1150     /* Free the volume list structures.
1151
1152     */
1153     if (NULL != rc->vol_list)
1154     {
1155         free(rc->vol_list);
1156     }
1157     /* Free the plugin related data
1158
1159     */
1160     if (NULL != rc->plugin_data)
1161     {
1162         (void) bvol_volidlist_destructor(
1163             rc->vol_list, BVOL_DESTROY_ASYN);
1164     }
1165 }
1166
1167 /* Free the backup app = 0;
1168    while (rc->currentFidT = rc->pliset)
1169    {
1170        rc->rc_backup_app++;
1171        rc->appData = rc->currentFidT->appData;
1172        /* Allow plugin to clean up and close .so: */
1173        if ( E_SUCCESS != err =
1174            rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
1175        {
1176            /* log error, continue */
1177            rc->user_error[ err ];
1178            RSTSL_Finish_failed_for_restore_plugin(
1179                (struct pluginData *) (
1180                    rc->currentFidT->idData )->name );
1181        }
1182        rc->pliset = rc->pliset->next;
1183    }
1184    rc->currentFidT = NULL;
1185 }
1186
1187 if (NULL != rc)
1188 {
1189     /* Free the multilevel structures */
1190     multilevel_cleanup(rc);
1191 }
1192 /* Free the mark bit map space
1193
1194 */
1195 for (mc_n = 0; mc_n < rc->rc_marks_plane_alloc; mc_n++)
1196 {
1197     if (NULL != rc->rc_marks[mc_n])
1198     {
1199         free(rc->rc_marks[mc_n]);
1200     }
1201     rc->rc_marks[mc_n] = NULL;
1202 }
1203 if (NULL != rc)
1204 {
1205     /* Free the backup app = 0;
1206        while (rc->currentFidT = rc->pliset)
1207        {
1208            rc->rc_backup_app++;
1209            rc->appData = rc->currentFidT->appData;
1210            /* Allow plugin to clean up and close .so: */
1211            if ( E_SUCCESS != err =
1212                rc->currentFidT->pluginArray[ rc->currentFidT->fidIndex ].pluginCleanup( rc ) )
1213            {
1214                /* log error, continue */
1215                rc->user_error[ err ];
1216                RSTSL_Finish_failed_for_restore_plugin(
1217                    (struct pluginData *) (
1218                        rc->currentFidT->idData )->name );
1219            }
1220            rc
```

Mon Oct 13 16:39:31 2008	RSTSL_Finish	Page 7 of 16
336 2)	
337 1	free (tcp->current_ptr);	
339 1	/*	
340 1	Free the various simple string buffers	
341 1	*/	
343 1	if (NULL != tcp->rc_top_level_object_name)	
344 2	free(tcp->rc_top_level_object_name);	
345 2		
346 1		
348 1	if (NULL != tcp->rc_template_name)	
349 2	free(tcp->rc_template_name);	
350 2		
351 1		
353 1	if (NULL != tcp->rc_workitem_name)	
354 2	free(tcp->rc_workitem_name);	
355 2		
356 1	if (NULL != tcp->rc_human_hostname)	
357 2	free(tcp->rc_human_hostname);	
358 2		
359 1		
361 1	if (NULL != tcp->rc_effective_username)	
362 2	{	
363 1	/* don't free, its internal: free(tcp->rc_effective_username);	
364 1	*/	
366 1	}	
368 1	if (NULL != tcp->rc_client_rbufname)	
369 2	{	
370 2	free(tcp->rc_client_rbufname);	
371 1	}	
373 1	if (NULL != tcp->rc_client_hostname)	
374 2	{	
375 2	free(tcp->rc_client_hostname);	
376 1	}	
378 1	if (NULL != tcp->rc_client_scriptname)	
379 2	{	
380 2	/* don't free, its internal: free(tcp->rc_client_scriptname);	
381 1	*/	
383 1	if (NULL != tcp->rc_client_dirtop)	
384 2	{	
385 2	free(tcp->rc_client_dirtop);	
386 1	}	
388 1	if (NULL != tcp->rc_cmd_context)	
389 2	{	
390 2	/* don't free -- its internal/temp data: free(
391 1	tcp->rc_cmd_context); */	
393 1	}	
394 1	if (NULL != tcp->rc_source_client_hostname)	
395 2	{	
396 2	free(tcp->rc_source_client_hostname);	
397 1	}	
399 1	if (NULL != tcp->rc_cplopen_executable)	
400 2	if (NULL != tcp->rc_cplopen_executable)	
401 1	*/	
403 1		
404 2	if (NULL != tcp->rc_plugin_wl_types)	
405 2	{	
406 1	free(tcp->rc_plugin_wl_types);	
407 2	}	
408 1	if (NULL != tcp->rc_pwd)	
409 2	{	
410 2	free(tcp->rc_pwd);	
411 1	}	
413 1	/*	
414 1	Finally, deallocate the xscore_context itself	
415 1	*/	
417 1	free(tcp);	
418 1	tcp = NULL;	
420 1	return(err);	
421 1	/* RSTSL_Finish */	

Mon Oct 13 16:39:31 2008	RSTSL_Finish	Page 8 of 16
399 2	{	
400 2	/* don't free, its internal: free(tcp->rc_cplopen_executable);	
401 1	*/	
403 1	}	
404 2	if (NULL != tcp->rc_plugin_wl_types)	
405 2	{	
406 1	free(tcp->rc_plugin_wl_types);	
407 2	}	
408 1	if (NULL != tcp->rc_pwd)	
409 2	{	
410 2	free(tcp->rc_pwd);	
411 1	}	
413 1	/*	
414 1	Finally, deallocate the xscore_context itself	
415 1	*/	
417 1	free(tcp);	
418 1	tcp = NULL;	
420 1	return(err);	
421 1	/* RSTSL_Finish */	

```

.....
// .....
int plugins;

Function Description:

This function locates, opens, validates and initializes all restore
plug-in (shared) libraries. They must be located in
/usr/apoc/RS/cure/plugin !
directory are opened and validated for version and presence or all
mandatory functions.
The RSMP_Identity function is called for each
library to determine which optional features are supported,
and those
the corresponding functions are present. Finally
the RSMP_InitLibrary
function is called for each valid library.

Parameters:

Inputs:
tcp (I) - Pointer to restore context

Outputs:
none

Returns:
E_SUCCESS or E_RSMP_RECOVER_ERR

Logic/pseudo code:

open plugin dir
while read_next_entry succeeds
verify so it is not a symlink
if it is a regular file (else continue)
on errors below:
close shared library file
continue
fetch all mandatory function addresses
call Identity function
check if it is a valid library
fetch all indicated optional function address
call Initialize function
add workitem types to composite exclusion list
add to valid plugin list
close plugin dir

/*
static errno_t Init_Plugins( restore_context *tcp )
{
DIR
struct dirent
struct stat
struct pluginData *pluginData = NULL;
int val_result;
struct pluginData *pluginData;
char
child_diren;
child_path [MAXPATHLEN];
}

```

```

484 1         open plugin_dir( &cur_plugin_dir ) )
485 1     {
486 1         if ( NULL == (dirp = opendir( eb_cur_plugin_dir )) )
487 1             return E_FAIL;
488 1         rec_ad_loc.com( SUB_CSM_PLUGIN_ERR, NULL );
489 1     }
490 2     #if 1
491 2         return E_SUCCESS; // allow continuation w/o plugins */
492 2     #else
493 2         return EP_RECOVER_NO_PLUGINS; // later do this */
494 2     #endif
495 1     struct shlib_path eb_cur_plugin_dir ;
496 1     struct shlib_path * pshlib_path ;
497 1     shlib_dirname = strmem( shlib_path );
498 1     /* Loop thru entries in directory */
499 1     while (NULL != (dirp = readdir( dirp )))
500 1     {
501 2         if ( NULL == strcmp( dirp->d_name, "." ) )
502 2             continue;
503 2         if ( NULL == strcmp( dirp->d_name, ".." ) )
504 2             continue;
505 2         if ( NULL == (plugin_desc = calloc( 1, sizeof(
506 2             struct plugin_descs ) ) ) )
507 2             break;
508 2         status = EP_RECOVER_WORKING;
509 2         /* fail thru to cleanup */
510 2     }
511 2     if ( NULL == strcmp( dirp->d_name, ".so*" ) )
512 2         continue; // skip this guy */
513 2     strcpy( shlib_path+shlib_dirname, dirp->d_name );
514 2     if ( NULL == (plugin_desc=>libpath)
515 2         || !dlopen( shlib_path, RTLD_NOW ) )
516 2     {
517 3         rbe_user_error( 0,
518 3             "Error opening restore plug-in library '%s' %s",
519 3             dirp->d_name, strerror( errno ) );
520 3         continue; // skip this one */
521 3     }
522 3     /* Pech addresses of all mandatory functions and */
523 3     /* Do identically processing: call it, save options, validate */
524 3     if ( 0 != (val_result = validate_plugin(
525 3         shlib_path, plugin_desc ) ) )
526 3     {
527 4         rbe_user_error( 0,
528 4             "Functions missing from restore plug-in library '%s'. Validation failed for restore plug-in library '%s'",
529 4             dirp->d_name );
530 4     }
531 4     else if ( val_result < 0 )
532 4     {
533 4         rbe_user_error( 0,
534 4             "Validation failed for restore plug-in library '%s'",
535 4             dirp->d_name );
536 4     }
537 4     else if ( val_result > 0 )
538 4     {
539 4         rbe_user_error( 0,
540 4             "Restore plug-in library '%s' is already loaded.",
541 4             dirp->d_name );

```

```

542 3         else
543 4             {
544 5                 /* the user_error( val_result,
545 6                     "RSPPL identity failed for restore plug-in library
546 7                     %s",
547 8                     dtemp->d_name );
548 9             }
549 10         fclose( pldataPcr->libhndl );
550 11         pldataPcr->libhndl = NULL; /* close .so on errors */
551 12         continue; /* on any error, skip this lib */
552 13     }
553 14 }
554 15
555 16 /* let DC plug-in do its initialization */
556 17 rcp->appdata = NULL;
557 18 /* enter plugin with clean appdata */
558 19 status =
559 20     pldataPcr->pfFuncarray[pFuncIndexInitialise]( rcp );
560 21 if ( E_SUCCESS != status )
561 22 {
562 23     /* the user_error( status,
563 24         "RSPPL_initialize failed for restore plug-in library
564 25         %s",
565 26         dtemp->d_name );
566 27     }
567 28     dtemp->d_name = NULL;
568 29     pldataPcr->libhndl = NULL;
569 30     status = E_SUCCESS; /* this was't fatal */
570 31     continue; /* on any error, skip this lib */
571 32 }
572 33
573 34 /* save plugin's appdata */
574 35 pldataPcr->appdata = rcp->appdata;
575 36 rcp->appdata = NULL;
576 37
577 38 /* add pldataPcr to valid plugin list */
578 39 if ( (NULL == pldataPcr)
579 40     || (NULL == pldataPcr) )
580 41     rcp->plist = pldataPcr; /* first in list */
581 42 else
582 43     pldataPcr->next = pldataPcr;
584 44 pldataPcr = pldataPcr; /* link from prev */
585 45
586 46 /* new end of list */
587 47 pldataPcr = NULL;
588 48
589 49 /* add specific types to composite exclusion list */
590 50 if ( pldataPcr == (struct pluginData *) pldataPcr->idbdata )
591 51 {
592 52     tmp_types = calloc( 1, 1 * idbdataPcr->num_types
593 53         + rcp->rc_num_plugin_wl_types );
594 54     if ( (NULL == tmp_types) )
595 55     {
596 56         status = EP_RB_RECOVER_NOMEM;
597 57         break;
598 58     }
599 59     if ( (NULL != rcp->rc_plugin_wl_types) )
600 60     {
601 61         /* move old list to new buffer and free old list */
602 62         memory( tmp_types,
603 63             rcp->rc_num_plugin_wl_types );
604 64         rcp->rc_num_plugin_wl_types = tmp_types;
605 65     }
606 66 }
607 67
608 68 /* free up leftovers */
609 69 if ( (NULL != pldataPcr) )
610 70     free( pldataPcr );
611 71
612 72 if ( E_SUCCESS != status )
613 73 {
614 74     /* Free contents of plugin list: */
615 75     while ( (NULL != (pluginData * pldataPcr)) )
616 76     {
617 77         /* also pluginData = pldataPcr->appdata;
618 78         rcp->appdata = pldataPcr->appdata;
619 79         pldataPcr->pfFuncarray[pFuncIndexInitialise](
620 80             rcp );
621 81         fclose( pldataPcr->libhndl );
622 82         pldataPcr = pldataPcr->next;
623 83         free( pldataPcr );
624 84     }
625 85     return status;

```

```

359 86     }
360 87     memory( tmp_types + rcp->rc_num_plugin_wl_types,
361 88         pldataPcr->num_types );
362 89     rcp->rc_num_plugin_wl_types +=
363 90         pldataPcr->num_types;
364 91     tmp_types[rcp->rc_num_plugin_wl_types] = 0;
365 92     rcp->rc_plugin_wl_types = tmp_types;
366 93 }
367 94
368 95 void closeLib( dirp );
369 96
370 97 /* Free up leftovers */
371 98 if ( (NULL != pldataPcr) )
372 99     free( pldataPcr );
373 100
374 101 if ( E_SUCCESS != status )
375 102 {
376 103     /* Free contents of plugin list: */
377 104     while ( (NULL != (pluginData * pldataPcr)) )
378 105     {
379 106         /* also pluginData = pldataPcr->appdata;
380 107         rcp->appdata = pldataPcr->appdata;
381 108         pldataPcr->pfFuncarray[pFuncIndexInitialise](
382 109             rcp );
383 110         fclose( pldataPcr->libhndl );
384 111         pldataPcr = pldataPcr->next;
385 112         free( pldataPcr );
386 113     }
387 114     return status;

```



```

626
627      /* init_plugins */
628      .....
629      validate_plugin
630      .....
631      *
632      * Function Description:
633      *
634      * This function retrieves the addresses of the mandatory plugin
635      * and stores them in the function pointer array,
636      * if any function is missing
637      *
638      * It returns -1.
639      *
640      * It then calls the identify_function and verifies wthe plugin
641      * version,
642      * and finds its optional functions. Specific error values are
643      * returned on version mismatch and missing optional functions.
644      *
645      * Parameters:
646      *
647      * Inputs:
648      *     plugin_ptr (
649      *         If - pointer to plugin data structure with libid1 set
650      *
651      * Outputs:
652      *     plugin_ptr in plugin_ptr is loaded with pointers to plugin
653      *         functions
654      *
655      * Returns:
656      *     0 on success
657      *     -1 on any missing required functions
658      *     -2 if version validation fails OR identify returns junk
659      *     -3 if worstcase type validation fails
660      *     -4 on any missing optional functions indicated by options
661      *
662      * Flags
663      *
664      * "n"
665      *     EG_RECOVER_XXX for error codes returned from identify function
666      *
667      .....
668      .....
669      static int validate_plugin( struct plugin_data *plugin_ptr )
670      {
671          int         index;
672          enum_ty      status;
673          struct plugin_data *iddata_ptr;
674
675          for( index = 0; index <= PIFUNCINDEXLASTBASIC; index++ )
676          {
677              if ( NULL == (plugin_ptr->pfncatarray[index]
678                          = (pfncat_ptr) dlopen( plugin_ptr->libid1,
679                                                  PIFUNCINDEX[index]
680                                                  ) ) )
681              {
682                  return -1;
683              }
684          }
685
686          /* call identify and validate */
687          status = validate_plugin_ptr(plugin_ptr->pfncatarray[index]);
688
689          if (status != E_SUCCESS)
690              return status;
691
692          if ( NULL == (iddata_ptr = {
693              struct plugin_data *plugin_ptr->iddata )
694              return -2;
695          }

```

Mon Oct 13 16:38:31 2008	valdiaz_dj@unl	Page 14 of 16
680 1	if (isDataPtr->version != RSPT_VERSION)	
680 2	{ /* only version 1 supported so far */	
680 3	pDataPtr->idData = NULL;	
680 4	return -2;	
680 5	}	
681 1	if (isDataPtr->numTypes && !isDataPtr->xtypes)	
681 2	{	
681 3	pDataPtr->idData = new unsigned short [null_pointer *]	
681 4	return -3;	
681 5	}	
682 1	/* if startAddress option set, get its addr or base */	
682 2	if ((RSPT_OPTION_SPECIAL_START & RSPT_OPTION_MASK_START))	
682 3	&& (NULL != isDataPtr->options & RSPT_OPTION_MASK_START)	
682 4	pDataPtr->idFuncAddr = pFuncIndex->startAddress	
682 5	= (pFuncPtr) delay(pDataPtr->libId,	
683 1	pFuncNames[pFuncIndex->startAddress])	
683 2	}	
683 3	/* OR if special find option set, get its addr or base */	
683 4	if ((RSPT_OPTION_SPECIAL_FIND	
683 5	&& (NULL != isDataPtr->options & RSPT_OPTION_MASK_FIND))	
684 1	&& (NULL != isDataPtr->idFuncAddr & pFuncIndex->id	
684 2	= (pFuncPtr) delay(pDataPtr->libId,	
684 3	pFuncNames[pFuncIndex->id])	
684 4	if (NULL == {	
684 5	pDataPtr->idFuncAddr = pFuncIndex->startAddress	
685 1	= (pFuncPtr) delay(pDataPtr->libId,	
685 2	pFuncNames[pFuncIndex->startAddress])	
685 3	if (NULL == {	
685 4	pDataPtr->idFuncAddr = RSPT_OPTION_MASK_GET_META)	
685 5	== isDataPtr->options & RSPT_OPTION_MASK_GET_META)	
686 1	&& (NULL == isDataPtr->idFuncAddr & pFuncIndex->id	
686 2	= (pFuncPtr) delay(pDataPtr->libId,	
686 3	pFuncNames[pFuncIndex->id])	
686 4	return 0;	
686 5	}	
687 1		
687 2		
687 3		
687 4		
687 5		
688 1		
688 2		
688 3		
688 4		
688 5		
689 1		
689 2		
689 3		
689 4		
689 5		
690 1		
690 2		
690 3		
690 4		
690 5		
691 1		
691 2		
691 3		
691 4		
691 5		
692 1		
692 2		
692 3		
692 4		
692 5		
693 1		
693 2		
693 3		
693 4		
693 5		
694 1		
694 2		
694 3		
694 4		
694 5		
695 1		
695 2		
695 3		
695 4		
695 5		
696 1		
696 2		
696 3		
696 4		
696 5		
697 1		
697 2		
697 3		
697 4		
697 5		
698 1		
698 2		
698 3		
698 4		
698 5		
699 1		
699 2		
699 3		
699 4		
699 5		
700 1		
700 2		
700 3		
700 4		
700 5		
701 1		
701 2		
701 3		
701 4		
701 5		
702 1		
702 2		
702 3		
702 4		
702 5		
703 1		
703 2		
703 3		
703 4		
703 5		
704 1		
704 2		
704 3		
704 4		
704 5		
705 1		
705 2		
705 3		
705 4		
705 5		
706 1		
706 2		
706 3		
706 4		
706 5		
707 1		
707 2		
707 3		
707 4		
707 5		
708 1		
708 2		
708 3		
708 4		
708 5		
709 1		
709 2		
709 3		
709 4		
709 5		
710 1		
710 2		
710 3		
710 4		
710 5		
711 1		
711 2		
711 3		
711 4		
711 5		
712 1		
712 2		
712 3		
712 4		
712 5		
713 1		
713 2		
713 3		
713 4		
713 5		
714 1		
714 2		
714 3		
714 4		
714 5		
715 1		
715 2		
715 3		
715 4		
715 5		
716 1		
716 2		
716 3		
716 4		
716 5		
717 1		
717 2		
717 3		
717 4		
717 5		
718 1		
718 2		
718 3		
718 4		
718 5		
719 1		
719 2		
719 3		
719 4		
719 5		
720 1		
720 2		
720 3		
720 4		
720 5		
721 1		
721 2		
721 3		
721 4		
721 5		
722 1		
722 2		
722 3		
722 4		
722 5		

Mon Oct 13 16:38:31 2008

RSMTimatic 14

Page 14 of 16

723

/*

validate_django */

```

1  /*
2  .....
3  1  *   File Name:   RSPplugin.h
4  2
5  3  *   Copyright (c) 1999 by BMC Corporation.
6  4
7  5  *   Purpose:
8  6
9  7  *       Two-fold:
10 8
11 9  *       - Define prototypes of functions in a restore plugin
12 10  *         - Define prototypes of common functions exported from
13 11  *           the
14 12  *             restore engine for use by restore plugin libraries.
15 13  *             These
16 14  *             functions are part of the restore_legacy library.
17 15
18 16  *   This header defines the API that a backup application must
19 17  *     be compatible with the new Restore API.
20 18
21 19  *   client-server based restore functionality formerly provided
22 20  *     RECOVER.API. The plug-in API is provided in a shared library
23 21  *     designed
24 22  *     to be called by the Restore Service Library -- the generic
25 23  *       server side
26 24  *       component of the Restore API that executes as part of the
27 25  *         Restore
28 26  *         Engine (server).
29 27
30 28  *   Compile-time options:
31 29
32 30  *   #define _POSIX_SOURCE 1
33 31  *   Need header files
34 32  *     #include <unistd.h>
35 33  *     #include <restore/api.h>
36 34  *     #include <restore/restorec.h>
37 35
38 36  *   #ifndef H_RSPAPI
39 37  *   #define H_RSPAPI
40 38  *   #endif
41 39
42 39  *   #include <extern "C" {
43 40
44 40  *   #include <osutil/osutil.h>
45 41  *   #include <error/e_db.h>
46 42  *   #include <restore/api.h>
47 43  *   #include <restore/restorec.h>
48 44
49 44  /* The following definition identifies the version of the Restore
50 45  *   plugin
51 46  *   header that a plug-in library was built with.
52 47  *   *   prototype changes, or a plugin function is added,
53 48  *     its value should be
54 49  *     updated. It will be used by the restore engine code to verify
55 50  *     compatibility with individual restore plugin libraries.
56 51  */
57 52
58 53
59 54
60 55
61 56
62 57
63 58
64 59
65 60
66 61
67 62
68 63
69 64
70 65
71 66
72 67
73 68
74 69
75 70
76 71
77 72
78 73
79 74
80 75
81 76
82 77
83 78
84 79
85 80
86 81
87 82
88 83
89 84
90 85
91 86
92 87
93 88
94 89
95 90
96 91
97 92
98 93
99 94
100 95
101 96
102 97
103 98
104 99
105 100
106 101
107 102
108 103
109 104
110 105
111 106
112 107
113 108
114 109
115 110
116 111
117 112
118 113
119 114
120 115
121 116
122 117
123 118
124 119
125 120
126 121
127 122
128 123
129 124
130 125
131 126
132 127
133 128
134 129
135 130
136 131
137 132
138 133
139 134
140 135
141 136
142 137
143 138
144 139
145 140
146 141
147 142
148 143
149 144
150 145
151 146
152 147
153 148
154 149
155 150
156 151
157 152
158 153
159 154
160 155
161 156
162 157
163 158
164 159
165 160
166 161
167 162
168 163
169 164
170 165
171 166
172 167
173 168
174 169
175 170
176 171
177 172
178 173
179 174
180 175
181 176
182 177
183 178
184 179
185 180
186 181
187 182
188 183
189 184
190 185
191 186
192 187
193 188
194 189
195 190
196 191
197 192
198 193
199 194
200 195
201 196
202 197
203 198
204 199
205 200
206 201
207 202
208 203
209 204
210 205
211 206
212 207
213 208
214 209
215 210
216 211
217 212
218 213
219 214
220 215
221 216
222 217
223 218
224 219
225 220
226 221
227 222
228 223
229 224
230 225
231 226
232 227
233 228
234 229
235 230
236 231
237 232
238 233
239 234
240 235
241 236
242 237
243 238
244 239
245 240
246 241
247 242
248 243
249 244
250 245
251 246
252 247
253 248
254 249
255 250
256 251
257 252
258 253
259 254
260 255
261 256
262 257
263 258
264 259
265 260
266 261
267 262
268 263
269 264
270 265
271 266
272 267
273 268
274 269
275 270
276 271
277 272
278 273
279 274
280 275
281 276
282 277
283 278
284 279
285 280
286 281
287 282
288 283
289 284
290 285
291 286
292 287
293 288
294 289
295 290
296 291
297 292
298 293
299 294
300 295
301 296
302 297
303 298
304 299
305 300
306 301
307 302
308 303
309 304
310 305
311 306
312 307
313 308
314 309
315 310
316 311
317 312
318 313
319 314
320 315
321 316
322 317
323 318
324 319
325 320
326 321
327 322
328 323
329 324
330 325
331 326
332 327
333 328
334 329
335 330
336 331
337 332
338 333
339 334
340 335
341 336
342 337
343 338
344 339
345 340
346 341
347 342
348 343
349 344
350 345
351 346
352 347
353 348
354 349
355 350
356 351
357 352
358 353
359 354
360 355
361 356
362 357
363 358
364 359
365 360
366 361
367 362
368 363
369 364
370 365
371 366
372 367
373 368
374 369
375 370
376 371
377 372
378 373
379 374
380 375
381 376
382 377
383 378
384 379
385 380
386 381
387 382
388 383
389 384
390 385
391 386
392 387
393 388
394 389
395 390
396 391
397 392
398 393
399 394
400 395
401 396
402 397
403 398
404 399
405 400
406 401
407 402
408 403
409 404
410 405
411 406
412 407
413 408
414 409
415 410
416 411
417 412
418 413
419 414
420 415
421 416
422 417
423 418
424 419
425 420
426 421
427 422
428 423
429 424
430 425
431 426
432 427
433 428
434 429
435 430
436 431
437 432
438 433
439 434
440 435
441 436
442 437
443 438
444 439
445 440
446 441
447 442
448 443
449 444
450 445
451 446
452 447
453 448
454 449
455 450
456 451
457 452
458 453
459 454
460 455
461 456
462 457
463 458
464 459
465 460
466 461
467 462
468 463
469 464
470 465
471 466
472 467
473 468
474 469
475 470
476 471
477 472
478 473
479 474
480 475
481 476
482 477
483 478
484 479
485 480
48
```

```

56 1 // Definitions of the indices of the plugin function addrs in the
57 1 // *of the plugindata structure.
58 1
59 1
60 1 // Mandatory functions */
61 1 #define PFuncIndexIdentify 0
62 1 #define PFuncIndexInit 1
63 1 #define PFuncIndexGetInfo 2
64 1 #define PFuncIndexGetCpuID 3
65 1 #define PFuncIndexGetCPU 4
66 1 #define PFuncIndexGetPCID 5
67 1 #define PFuncIndexGetNIC 6
68 1 #define PFuncIndexGetHost 7
69 1 #define PFuncIndexGetOSRelease 8
70 1 #define PFuncIndexGetArch 9
71 1 #define PFuncIndexGetMach 10
72 1 #define PFuncIndexIsMarked 11
73 1 #define PFuncIndexIsMarkedBy 12
74 1 #define PFuncIndexGetAllItems 13
75 1 #define PFuncIndexGetItem 14
76 1 #define PFuncIndexGetPrioItem 15
77 1 #define PFuncIndexGetPreViewTime 16
78 1 #define PFuncIndexGetNextTime 17
79 1 #define PFuncIndexGetTestTime 18
80 1 #define PFuncIndexGetServerCnt 19
81 1 #define PFuncIndexGetServerIP 20
82 1 #define PFuncIndexGetServerPort 21
83 1 #define PFuncIndexGetServerPortList 22
84 1 #define PFuncIndexGetServerPortListLen 23
85 1 #define PFuncIndexGetFiniish 24
86 1 #define PFuncIndexGetLastIsolic 24
87 1
88 1 // Optional functions: */
89 1 #define PFuncIndexGetStateScore 25
90 1 #define PFuncIndexFind 26
91 1 #define PFuncIndexFindResults 27
92 1 #define PFuncIndexGetSetCPU 28
93 1 #define PFuncIndexGetLast 28
94 1
95 1 // Definitions of the names of the plugin functions in the pifuncarray
96 1 // * of the plugindata structure.
97 1 // These must be in the same order and position
98 1 // as the pifuncindex values above,
99 1 // and are initialized in RSInitInfc.c.
100 1 char pifuncNames[PFuncIndexLast+1];
101 1
102 1 typedef enum_t { /* generic plugin func prototype */
103 1     typedef struct restore_context restore_context;
104 1 } /* for function prototypes */
105 1
106 1 // callback function prototypes: */
107 1 typedef bool_main_ty (*ASRPI_MarkProgressProc) (
108 1     unsigned long totalMarks);
109 1 typedef bool_main_ty (*ASRPI_SubmitProgressProc) (
110 1     unsigned long totalElements);

```



```

212 1      * This function is called once, to allow the plug-in to perform its
213 1      * local initialization.
214 1      * The restore context will have already been
215 1      * initialized with the config file parsed,
216 1      * and the human user fields
217 1      * set.
218 1      *
219 1      * Parameters:
220 1      * context (I) - Pointer to the restore context
221 1      *
222 1      * Returns:
223 1      * E_SUCCESS on success
224 1      * E_IO_RECOVER_XXX on error
225 1      *
226 1      * =====
227 1      * =====
228 1      * =====
229 1      * =====
230 1      * =====
231 1      * =====
232 1      * =====
233 1      * This function is called to allow the plug-in to clean up its
234 1      * internal
235 1      * storage when a restore session is ending.
236 1      *
237 1      * Parameters:
238 1      * context (I) - Pointer to the restore context
239 1      *
240 1      * Returns:
241 1      * E_SUCCESS on success
242 1      * E_IO_RECOVER_XXX on error
243 1      *
244 1      * =====
245 1      * =====
246 1      * =====
247 1      * =====
248 1      * =====
249 1      * =====
250 1      * =====
251 1      * =====
252 1      * =====
253 1      * =====
254 1      * =====
255 1      * =====
256 1      * =====
257 1      * =====
258 1      * =====
259 1      * =====
260 1      * =====
261 1      * =====
262 1      * =====
263 1      * =====
264 1      * =====
265 1      * =====
266 1      * =====
267 1      * =====
268 1      * =====
269 1      * =====
270 1      * =====
271 1      * =====
272 1      * =====
273 1      * =====
274 1      * =====
275 1      * =====
276 1      * =====
277 1      * =====
278 1      * =====
279 1      * =====
280 1      * =====
281 1      * =====
282 1      * =====
283 1      * =====
284 1      * =====
285 1      * =====
286 1      * =====
287 1      * =====
288 1      * =====
289 1      * =====
290 1      * =====
291 1      * =====
292 1      * =====
293 1      * =====
294 1      * =====
295 1      * =====
296 1      * =====
297 1      * =====
298 1      * =====
299 1      * =====
300 1      * =====
301 1      * =====
302 1      * =====
303 1      * =====
304 1      * =====
305 1      * =====
306 1      * =====
307 1      * =====
308 1      * =====
309 1      * =====
310 1      * =====
311 1      * =====
312 1      * =====
313 1      * =====
314 1      * =====
315 1      * =====
316 1      * =====
317 1      * =====
318 1      * =====
319 1      * =====
320 1      * =====
321 1      * =====
322 1      * =====
323 1      * =====
324 1      * =====
325 1      * =====
326 1      * =====
327 1      * =====
328 1      * =====
329 1      * =====
330 1      * =====
331 1      * =====
332 1      * =====
333 1      * =====
334 1      * =====
335 1      * =====
336 1      * =====
337 1      * =====
338 1      * =====
339 1      * =====
340 1      * =====
341 1      * =====
342 1      * =====
343 1      * =====
344 1      * =====
345 1      * =====
346 1      * =====
347 1      * =====
348 1      * =====
349 1      * =====
350 1      * =====
351 1      * =====
352 1      * =====
353 1      * =====
354 1      * =====
355 1      * =====
356 1      * =====
357 1      * =====
358 1      * =====
359 1      * =====
360 1      * =====
361 1      * =====
362 1      * =====
363 1      * =====
364 1      * =====
365 1      * =====
366 1      * =====
367 1      * =====
368 1      * =====
369 1      * =====
370 1      * =====
371 1      * =====
372 1      * =====
373 1      * =====
374 1      * =====
375 1      * =====
376 1      * =====
377 1      * =====
378 1      * =====
379 1      * =====
380 1      * =====
381 1      * =====
382 1      * =====
383 1      * =====
384 1      * =====
385 1      * =====
386 1      * =====
387 1      * =====
388 1      * =====
389 1      * =====
390 1      * =====
391 1      * =====
392 1      * =====
393 1      * =====
394 1      * =====
395 1      * =====
396 1      * =====
397 1      * =====
398 1      * =====
399 1      * =====
400 1      * =====
401 1      * =====
402 1      * =====
403 1      * =====
404 1      * =====
405 1      * =====
406 1      * =====
407 1      * =====
408 1      * =====
409 1      * =====
410 1      * =====
411 1      * =====
412 1      * =====
413 1      * =====
414 1      * =====
415 1      * =====
416 1      * =====
417 1      * =====
418 1      * =====
419 1      * =====
420 1      * =====
421 1      * =====
422 1      * =====
423 1      * =====
424 1      * =====
425 1      * =====
426 1      * =====
427 1      * =====
428 1      * =====
429 1      * =====
430 1      * =====
431 1      * =====
432 1      * =====
433 1      * =====
434 1      * =====
435 1      * =====
436 1      * =====
437 1      * =====
438 1      * =====
439 1      * =====
440 1      * =====
441 1      * =====
442 1      * =====
443 1      * =====
444 1      * =====
445 1      * =====
446 1      * =====
447 1      * =====
448 1      * =====
449 1      * =====
450 1      * =====
451 1      * =====
452 1      * =====
453 1      * =====
454 1      * =====
455 1      * =====
456 1      * =====
457 1      * =====
458 1      * =====
459 1      * =====
460 1      * =====
461 1      * =====
462 1      * =====
463 1      * =====
464 1      * =====
465 1      * =====
466 1      * =====
467 1      * =====
468 1      * =====
469 1      * =====
470 1      * =====
471 1      * =====
472 1      * =====
473 1      * =====
474 1      * =====
475 1      * =====
476 1      * =====
477 1      * =====
478 1      * =====
479 1      * =====
480 1      * =====
481 1      * =====
482 1      * =====
483 1      * =====
484 1      * =====
485 1      * =====
486 1      * =====
487 1      * =====
488 1      * =====
489 1      * =====
490 1      * =====
491 1      * =====
492 1      * =====
493 1      * =====
494 1      * =====
495 1      * =====
496 1      * =====
497 1      * =====
498 1      * =====
499 1      * =====
500 1      * =====
501 1      * =====
502 1      * =====
503 1      * =====
504 1      * =====
505 1      * =====
506 1      * =====
507 1      * =====
508 1      * =====
509 1      * =====
510 1      * =====
511 1      * =====
512 1      * =====
513 1      * =====
514 1      * =====
515 1      * =====
516 1      * =====
517 1      * =====
518 1      * =====
519 1      * =====
520 1      * =====
521 1      * =====
522 1      * =====
523 1      * =====
524 1      * =====
525 1      * =====
526 1      * =====
527 1      * =====
528 1      * =====
529 1      * =====
530 1      * =====
531 1      * =====
532 1      * =====
533 1      * =====
534 1      * =====
535 1      * =====
536 1      * =====
537 1      * =====
538 1      * =====
539 1      * =====
540 1      * =====
541 1      * =====
542 1      * =====
543 1      * =====
544 1      * =====
545 1      * =====
546 1      * =====
547 1      * =====
548 1      * =====
549 1      * =====
550 1      * =====
551 1      * =====
552 1      * =====
553 1      * =====
554 1      * =====
555 1      * =====
556 1      * =====
557 1      * =====
558 1      * =====
559 1      * =====
560 1      * =====
561 1      * =====
562 1      * =====
563 1      * =====
564 1      * =====
565 1      * =====
566 1      * =====
567 1      * =====
568 1      * =====
569 1      * =====
570 1      * =====
571 1      * =====
572 1      * =====
573 1      * =====
574 1      * =====
575 1      * =====
576 1      * =====
577 1      * =====
578 1      * =====
579 1      * =====
580 1      * =====
581 1      * =====
582 1      * =====
583 1      * =====
584 1      * =====
585 1      * =====
586 1      * =====
587 1      * =====
588 1      * =====
589 1      * =====
590 1      * =====
591 1      * =====
592 1      * =====
593 1      * =====
594 1      * =====
595 1      * =====
596 1      * =====
597 1      * =====
598 1      * =====
599 1      * =====
600 1      * =====
601 1      * =====
602 1      * =====
603 1      * =====
604 1      * =====
605 1      * =====
606 1      * =====
607 1      * =====
608 1      * =====
609 1      * =====
610 1      * =====
611 1      * =====
612 1      * =====
613 1      * =====
614 1      * =====
615 1      * =====
616 1      * =====
617 1      * =====
618 1      * =====
619 1      * =====
620 1      * =====
621 1      * =====
622 1      * =====
623 1      * =====
624 1      * =====
625 1      * =====
626 1      * =====
627 1      * =====
628 1      * =====
629 1      * =====
630 1      * =====
631 1      * =====
632 1      * =====
633 1      * =====
634 1      * =====
635 1      * =====
636 1      * =====
637 1      * =====
638 1      * =====
639 1      * =====
640 1      * =====
641 1      * =====
642 1      * =====
643 1      * =====
644 1      * =====
645 1      * =====
646 1      * =====
647 1      * =====
648 1      * =====
649 1      * =====
650 1      * =====
651 1      * =====
652 1      * =====
653 1      * =====
654 1      * =====
655 1      * =====
656 1      * =====
6
```

307	1	*/	numberBites (0) - the real number of objects returned in the list	
308	1				
309	1		Returns:	on success	
310	1		E_SUCCESS	on success	
311	1		EPLM_RECOVER_xxx	on error	
312	1				
313	1				
314	1				
315	1				
316	1				
317	1				
318	1				
319	1				
320	1				
321	1				
322	1				
323	1				
324	1				
325	1				
326	1				
327	1				
328	1				
329	1				
330	1				
331	1				
332	1				
333	1				
334	1				
335	1				
336	1				
337	1				
338	1				
339	1				
340	1				
341	1				
342	1				
343	1				
344	1				
345	1				
346	1				
347	1				
348	1				
349	1				
350	1				
351	1				
352	1				
353	1				
354	1				
355	1				
356	1				
357	1				
358	1				
359	1				
360	1				
361	1				
362	1				
363	1				
364	1				
365	1				
366	1				
367	1				
368	1				
369	1				
370	1				
371	1				
372	1				
373	1				
374	1				
375	1				
376	1				
377	1				
378	1				
379	1				
380	1				
381	1				
382	1				
383	1				
384	1				
385	1				
386	1				
387	1				
388	1				
389	1				
390	1				
391	1				
392	1				
393	1				
394	1				
395	1				
396	1				
397	1				
398	1				
399	1				
400	1				
401	1				
402	1				
403	1				
404	1				
405	1				
406	1				
407	1				
408	1				
409	1				
410	1				
411	1				
412	1				
413	1				
414	1				
415	1				
416	1				
417	1				
418	1				
419					

344	*	objects	{	- a pointer to receive the start of the objects list
345	*	cookie	(10) - a place holder for the list position	
346	*	maxEntries	(1) - the maximum number of objects to return	
347	*	numEntries	{ the total number of objects returned in the list	
348	*	allowHidden	(1) - flag whether or not to include bad files	
349	*	Return:		
350	*	R_SUCCESS	on success	
351	*	R_ERR_RECOVER_xxx	on error	
352	*			
353	*			
354	*			
355	*	errno_t	restore_context	
356	*	restoreObjDescr	parentObj	
357	*	attr	RSRNC_undo_list	
358	*	cookie	cookie	
359	*	long	long	
360	*	long	long	
361	*	maxEntries	maxEntries	
362	*	const boolean_t	allowHidden	
363	*			
364	*			
365	*			
366	*			
367	*			
368	*			
369	*			
370	*			
371	*			
372	*			
373	*			
374	*			
375	*			
376	*			
377	*			
378	*			
379	*			
380	*			

381	*	submitObjDescr(
382	*	IOI	- ID of the submit user object created to describe the restore
383	*	progress	{
384	*	IOI	- pointer to callback function to report progress and use for cancellation
385	*		
386	*	errno_t	RSRPL_Submit(
387	*	restore_context	
388	*	const char	hostname
389	*	const boolean_t	allowHidden
390	*	const boolean_t	allowHidden
391	*	const RestoreTransport	transport
392	*	int	RSRPL_SubmitProgressProc
393	*	progress	progress
394	*		
395	*		
396	*		
397	*		
398	*		
399	*		
400	*		
401	*		
402	*		
403	*		
404	*		
405	*		
406	*		
407	*		
408	*		
409	*		
410	*		
411	*		
412	*		
413	*		
414	*		
415	*		
416	*		
417	*		
418	*		
419	*		
420	*		
421	*		
422	*		
423	*		
424	*		
425	*		
426	*		
427	*		
428	*		
429	*		
430	*		
431	*		
432	*		

433	*	Find Routines:	
434	*	RSRPL_FindRestoreObjDescrs	
435	*	RSRPL_GetFindResults	
436	*		
437	*		
438	*		
439	*		
440	*		
441	*		
442	*		
443	*		
444	*		
445	*		
446	*		
447	*		
448	*		
449	*		
450	*		
451	*		
452	*		
453	*		
454	*		
455	*		
456	*		
457	*		
458	*		
459	*		
460	*		
461	*		
462	*		
463	*		
464	*		
465	*		
466	*		
467	*		
468	*		
469	*		
470	*		
471	*		
472	*		
473	*		
474	*		
475	*		
476	*		
477	*		
478	*		
479	*		
480	*		
481	*		
482	*		
483	*		
484	*		
485	*		
486	*		
487	*		
488	*		
489	*		
490	*		
491	*		
492	*		
493	*		
494	*		
495	*		
496	*		
497	*		
498	*		
499	*		
500	*		

540.1	struct RSTRC_user_restoreable_object *thisobject;	
541.1	boolean Ly allowedFiles;	
542.1	unsigned long *BackupCount;	
543.1	unsigned long *RemoveKeyCount;	
544.1	unsigned long *UnmarkedFiles;	
545.1	u_byter *UnmarkedFiles;	
546.1	unsigned long *dismarked;	
547.1	unsigned long *otherMarked;	
548.1	RSTRC_ProgressProc progressCB ;	
549.1	
550.1	
551.1	
552.1	
553.1	
554.1	
555.1	
556.1	
557.1	
558.1	
559.1	
560.1	
561.1	
562.1	
563.1	
564.1	
565.1	
566.1	
567.1	
568.1	
569.1	
570.1	
571.1	
572.1	
573.1	
574.1	
575.1	
576.1	
577.1	
578.1	
579.1	
580.1	
581.1	
582.1	
583.1	
584.1	
585.1	
586.1	
587.1	
588.1	
589.1	
590.1	
591.1	
592.1	
593.1	
594.1	
595.1	
596.1	
597.1	
598.1	
599.1	
600.1	
601.1	
602.1	
603.1	
604.1	
605.1	
606.1	
607.1	
608.1	
609.1	
610.1	
611.1	
612.1	
613.1	
614.1	
615.1	
616.1	
617.1	
618.1	
619.1	
620.1	
621.1	
622.1	
623.1	
624.1	
625.1	
626.1	
627.1	
628.1	
629.1	
630.1	
631.1	
632.1	
633.1	
634.1	
635.1	
636.1	
637.1	
638.1	
639.1	
640.1	
641.1	
642.1	
643.1	
644.1	
645.1	
646.1	
647.1	
648.1	
649.1	
650.1	
651.1	
652.1	
653.1	
654.1	
655.1	
656.1	
657.1	
658.1	
659.1	
660.1	
661.1	
662.1	
663.1	
664.1	
665.1	
666.1	
667.1	
668.1	
669.1	
670.1	
671.1	
672.1	
673.1	
674.1	
675.1	
676.1	
677.1	
678.1	
679.1	
680.1	
681.1	
682.1	
683.1	
684.1	
685.1	
686.1	
687.1	
688.1	
689.1	
690.1	
691.1	
692.1	
693.1	
694.1	
695.1	
696.1	
697.1	
698.1	
699.1	
700.1	

590.1	
591.1	
592.1	
593.1	
594.1	
595.1	
596.1	
597.1	
598.1	
599.1	
600.1	
601.1	
602.1	
603.1	
604.1	
605.1	
606.1	
607.1	
608.1	
609.1	
610.1	
611.1	
612.1	
613.1	
614.1	
615.1	
616.1	
617.1	
618.1	
619.1	
620.1	
621.1	
622.1	
623.1	
624.1	
625.1	
626.1	
627.1	
628.1	
629.1	
630.1	
631.1	
632.1	
633.1	
634.1	
635.1	
636.1	
637.1	
638.1	
639.1	
640.1	
641.1	
642.1	
643.1	
644.1	
645.1	
646.1	
647.1	
648.1	
649.1	
650.1	
651.1	
652.1	
653.1	
654.1	
655.1	
656.1	
657.1	
658.1	
659.1	
660.1	
661.1	
662.1	
663.1	
664.1	
665.1	
666.1	
667.1	
668.1	
669.1	
670.1	
671.1	
672.1	
673.1	
674.1	
675.1	
676.1	
677.1	
678.1	
679.1	
680.1	
681.1	
682.1	
683.1	
684.1	
685.1	
686.1	
687.1	
688.1	
689.1	
690.1	
691.1	
692.1	
693.1	
694.1	
695.1	
696.1	
697.1	
698.1	
699.1	
700.1	


```

443 1 * Parameters:
444 1 * context - (I) - Pointer to the restore context
445 1 * objects - (O) - addresses to return a pointer to the list of objects in
446 1 * numberlist (
447 1 * O) - the real number of media objects returned in the array
448 1 *
449 1 ****
450 1 restoreLy RSPT_GetMediaAsMedia() restore context *context, **objects,
451 1 short *numberlist );
452 1 ****
453 1
454 1 *
455 1 * Get All Backup Times
456 1 *
457 1 * Function Description:
458 1 * Retrieve the dates of the backups within the time range
459 1 * specified by the caller.
460 1 *
461 1 * Parameters:
462 1 * context - (I) - Pointer to the restore context
463 1 * starttime - (I) include no earlier than this date
464 1 * endtime - (I) include no later than this date
465 1 * flags
466 1 * I) Backup consisting flags: e.g. full-only/partial-ok
467 1 * O) per to linked list of times
468 1 * O) count of times returned
469 1 *
470 1 * Return Codes:
471 1 * E_SUCCESS - operation completed successfully
472 1 *
473 1 ****
474 1
475 1 restoreLy
476 1 RSPT_GetAllBackupTimes( restore context *context,
477 1 const time_t *starttime,
478 1 const time_t *endtime,
479 1 RSPT_Backup_Flags_t flags,
480 1 struct RSPT_time_list **timelist,
481 1 short *numberlist );
482 1 ****
483 1
484 1 *
485 1 * RSPT_GetCurrentBackupTime
486 1 *
487 1 * Function Description:
488 1 * Retrieve the time of the backup that the current restore
489 1 * is set to and return it in the preallocated buffer.
490 1 * context
491 1 *
492 1 * Parameters:
493 1 * context - (I) Pointer to the restore context
494 1 * backupTime - (O) the time of the backup
495 1 *
496 1 *
497 1 * Return Codes:
498 1 * E_SUCCESS - operation completed successfully
499 1 * E_INVALID_PARAMETER - call passed out of sequence
500 1 * E_RSPT_RECOVER_ALREADY - already recovered
501 1 * E_RSPT_RECOVER_NO_CURR_BACKUP - no valid backup currently
502 1 *
503 1 ****
504 1

```

```

703 | RSTPL_CurrentBackupTime() restore_context "context;"
704 | time_c "backupTime";
705 | /***** ** */
706 |
707 | * Set Backup For Time
708 |
709 | * Function Description:
710 |   switch to the backup plane of the specified time,
711 |   or the most recent
712 |   that is before the specified time,
713 |   if an exact match is not possible.
714 | * Parameters:
715 |   context (I) - pointer to the restore context
716 |   backup_time (I) - the time for which the backup is requested
717 |   flags (I) - backup constraint flags: e.g., full-only/partial-ok
718 | * Return Codes:
719 |   E_SUCCESS - operation completed successfully
720 |   E_RN_RECOVER_KXX - backup plane cannot be found
721 |   **
722 | /*****/
723 |
724 | extern_cy
725 | RSTPL_SetBackupForTime( restore_context
726 |                          const time_c
727 |                          "context,"
728 |                          RSTPL_Backup_Flags_t flags );
729 | /*****/
730 |
731 | * See Previous Backup
732 |
733 | * Function Description:
734 |   see the restore context to that of the previous backup with
735 |   respect
736 |   to the current one.
737 | * Parameters:
738 |   context (I) - pointer to the restore context
739 |   flags (I) - backup constraint flags: e.g., full-only/partial-ok
740 | * Return Codes:
741 |   E_SUCCESS - operation completed successfully
742 |   E_RN_NO PREV CATALOG - when at the first catalog
743 |   E_RN_ABOVE_PERMISSION_DENIED - when user cannot access the file
744 |   **
745 | of the new catalog
746 | *
747 | file
748 |
749 | /*****/
750 | extern_cy
751 | RSTPL_SetPrevBackup( restore_context
752 |                      RSTPL_Backup_Flags_t flags );
753 | /*****/
754 |
755 | * See Next Backup
756 |
757 | * Function Description:
758 |   this routine must set the restore environment to the the next
759 |   backup
760 | * of the current copy level object.

```

745.1	*	Parameters:	
745.2	*	Context (I) - Pointer to the restore context	
745.3	*	Flags	
745.4	*	I) - Backup constraint flags: e.g., full-only/partial-ok	
745.5	*	Return Codes:	
745.6	*	E_SUCCESS - operation completed successfully	
745.7	*	EP_RB_RECOVER_NO_MGMT_CATALOG - when at the most recent catalog	
745.8	*	EP_RB_RECOVER_PERMISSION_DENIED - when user cannot access the file	
745.9	*	of the new catalog	
745.10	*	EP_RB_RECOVER_NO_CATALOG - when mcat_set_moplane failed	
745.11	*	*****	
745.12	*	*****	
745.13	*	*****	
745.14	*	*****	
745.15	*	*****	
745.16	*	*****	
745.17	*	*****	
745.18	*	*****	
745.19	*	*****	
745.20	*	*****	
745.21	*	*****	
745.22	*	*****	
745.23	*	*****	
745.24	*	*****	
745.25	*	*****	
745.26	*	*****	
745.27	*	*****	
745.28	*	*****	
745.29	*	*****	
745.30	*	*****	
745.31	*	*****	
745.32	*	*****	
745.33	*	*****	
745.34	*	*****	
745.35	*	*****	
745.36	*	*****	
745.37	*	*****	
745.38	*	*****	
745.39	*	*****	
745.40	*	*****	
745.41	*	*****	
745.42	*	*****	
745.43	*	*****	
745.44	*	*****	
745.45	*	*****	
745.46	*	*****	
745.47	*	*****	
745.48	*	*****	
745.49	*	*****	
745.50	*	*****	
745.51	*	*****	
745.52	*	*****	
745.53	*	*****	
745.54	*	*****	
745.55	*	*****	
745.56	*	*****	
745.57	*	*****	
745.58	*	*****	
745.59	*	*****	
745.60	*	*****	
745.61	*	*****	
745.62	*	*****	
745.63	*	*****	
745.64	*	*****	
745.65	*	*****	
745.66	*	*****	
745.67	*	*****	
745.68	*	*****	
745.69	*	*****	
745.70	*	*****	
745.71	*	*****	
745.72	*	*****	
745.73	*	*****	
745.74	*	*****	
745.75	*	*****	
745.76	*	*****	
745.77	*	*****	
745.78	*	*****	
745.79	*	*****	
745.80	*	*****	
745.81	*	*****	
745.82	*	*****	
745.83	*	*****	
745.84	*	*****	
745.85	*	*****	
745.86	*	*****	
745.87	*	*****	
745.88	*	*****	
745.89	*	*****	
745.90	*	*****	
745.91	*	*****	
745.92	*	*****	
745.93	*	*****	
745.94	*	*****	
745.95	*	*****	
745.96	*	*****	
745.97	*	*****	
745.98	*	*****	
745.99	*	*****	
745.100	*	*****	
745.101	*	*****	
745.102	*	*****	
745.103	*	*****	
745.104	*	*****	
745.105	*	*****	
745.106	*	*****	
745.107	*	*****	
745.108	*	*****	
745.109	*	*****	
745.110	*	*****	
745.111	*	*****	
745.112	*	*****	
745.113	*	*****	
745.114	*	*****	
745.115	*	*****	
745.116	*	*****	
745.117	*	*****	
745.118	*	*****	
745.119	*	*****	
745.120	*	*****	
745.121	*	*****	
745.122	*	*****	
745.123	*	*****	
745.124	*	*****	
745.125	*	*****	
745.126	*	*****	
745.127	*	*****	
745.128	*	*****	
745.129	*	*****	
745.130	*	*****	
745.131	*	*****	
745.132	*	*****	
745.133	*	*****	
745.134	*	*****	
745.135	*	*****	
745.136	*	*****	
745.137	*	*****	
745.138	*	*****	
745.139	*	*****	
745.140	*	*****	
745.141	*	*****	
745.142	*	*****	
745.143	*	*****	
745.144	*	*****	
745.145	*	*****	
745.146	*	*****	
745.147	*	*****	
745.148	*	*****	
745.149	*	*****	
745.150	*	*****	
745.151	*	*****	
745.152	*	*****	
745.153	*	*****	
745.154	*	*****	
745.155	*	*****	
745.156	*	*****	
745.157	*	*****	
745.158	*	*****	
745.159	*	*****	
745.160	*	*****	
745.161	*	*****	
745.162	*	*****	
745.163	*	*****	
745.164	*	*****	
745.165	*	*****	
745.166	*	*****	
745.167	*	*****	
745.168	*	*****	
745.169	*	*****	
745.170	*	*****	
745.171	*	*****	
745.172	*	*****	
745.173	*	*****	
745.174	*	*****	
745.175	*	*****	
745.176	*	*****	
745.177	*	*****	
745.178	*	*****	
745.179	*	*****	
745.180	*	*****	
745.181	*	*****	
745.182	*	*****	
745.183	*	*****	
745.184	*	*****	
745.185	*	*****	
745.186	*	*****	
745.187	*	*****	
745.188	*	*****	
745.189	*	*****	
745.190	*	*****	
745.191	*	*****	
745.192	*	*****	
745.193	*	*****	
745.194	*	*****	
745.195	*	*****	
745.196	*	*****	
745.197	*	*****	
745.198	*	*****	
745.199	*	*****	
745.200	*	*****	
745.201	*	*****	
745.202	*	*****	
745.203	*	*****	
745.204	*	*****	
745.205	*	*****	
745.206	*	*****	
745.207	*	*****	
745.208	*	*****	
745.209	*	*****	
745.210	*	*****	
745.211	*	*****	
745.212	*	*****	
745.213	*	*****	
745.214	*	*****	
745.215	*	*****	
745.216	*	*****	
745.217	*	*****	
745.218	*	*****	
745.219	*	*****	
745.220	*	*****	
745.221	*	*****	
745.222	*	*****	
745.223	*	*****	
745.224	*	*****	
745.225	*	*****	
745.226	*	*****	
745.227	*	*****	
745.228	*	*****	
745.229	*	*****	
745.230	*	*****	
745.231	*	*****	
745.232	*	*****	
745.233	*	*****	
745.234	*	*****	
745.235	*	*****	
745.236	*	*****	
745.237	*	*****	
745.238	*	*****	
745.239	*	*****	
745.240	*	*****	
745.241	*	*****	
745.242	*	*****	
745.243	*	*****	
745.244	*	*****	
745.245	*	*****	
745.246	*	*****	
745.247	*	*****	
745.248	*	*****	
745.249	*	*****	
745.250	*	*****	
745.251	*	*****	
745.252	*	*****	
745.253	*	*****	
745.254	*	*****	
745.255	*	*****	
745.256	*	*****	
745.257	*	*****	
745.258	*	*****	
745.259	*	*****	
745.260	*	*****	
745.261	*	*****	
745.262	*	*****	
745.263	*	*****	
745.264	*	*****	
745.265	*	*****	
745.266	*	*****	
745.267	*	*****	
745.268	*	*****	
745.269	*	*****	
745.270	*	*****	
745.271	*	*****	
745.272	*	*****	
745.273	*	*****	
745.274	*	*****	
745.275	*	*****	
745.276	*	*****	
745.277	*	*****	
745.278	*	*****	
745.279	*	*****	
745.280	*	*****	
745.281	*	*****	
745.282	*	*****	
745.283	*	*****	
745.284	*	*****	
745.285	*	*****	
745.286	*	*****	
745.287	*	*****	
745.288	*	*****	
745.289	*	*****	
745.290	*	*****	
745.291	*	*****	
745.292	*	*****	
745.293	*	*****	
745.294	*	*****	
745.295	*	*****	
745.296	*	*****	
745.297	*	*****	
745.298	*	*****	
745.299	*	*****	
745.300	*	*****	
745.301	*	*****	
745.302	*	*****	
745.303	*	*****	
745.304	*	*****	
745.305	*	*****	
745.306	*	*****	
745.307	*	*****	
745.308	*	*****	
745.309	*	*****	
745.310	*	*****	
745.311	*	*****	
745.312	*	*****	
745.313	*	*****	
745.314	*	*****	
745.315	*	*****	
745.316	*	*****	
745.317	*	*****	
745.318	*	*****	
745.319	*	*****	
745.320	*	*****	
745.321	*	*****	
745.322	*	*****	
745.323	*	*****	
745.324	*	*****	
745.325	*	*****	
745.326	*	*****	
745.327	*	*****	
745.328	*	*****	
745.329	*	*****	
745.330	*	*****	
745.331	*	*****	
745.332	*	*****	
745.333	*	*****	
745.334	*	*****	
745.335	*	*****	
745.336	*	*****	
745.337	*	*****	
745.338	*	*****	
745.339	*	*****	
745.340	*	*****	
745.341	*	*****	
745.342	*	*****	
745.343	*	*****	
745.344	*	*****	
745.345	*	*****	
745.346	*	*****	
745.347	*	*****	
745.348	*	*****	
745.349	*	*****	
745.350	*	*****	
745.351	*	*****	
745.352	*	*****	
745.353	*	*****	
745.354	*	*****	
745.355	*	*****	
745.356	*	*****	
745.357	*	*****	
745.358	*	*****	
745.359	*	*****	
745.360	*	*****	
745.361	*	*****	
745.362	*	*****	

```

874 1      * Function Description:
875 1      * Determine if a backup exists prior to the specified time
876 1
877 1
878 1      * Parameters:
879 1      * context (I) - Pointer to the restore context
880 1      * thisTime(I) - Time for the query
881 1      * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
882 1      * isThere (O) - TRUE/FALSE that requested backup does exist
883 1
884 1      * Return Codes:
885 1      * E_SUCCESS - operation completed successfully
886 1      * EP_RE_RECOVER_xxx - when errors occur accessing catalogs
887 1
888 1      *
889 1
890 1      restore_ly
891 1      RSTPL_IsTherePrevBackupForTime( restore_context
892 1      *context,
893 1      *context,
894 1      RSTPL_Backup_flags_ly
895 1      flags,
896 1      boolean_ly
897 1      );
898 1
899 1      /*****
900 1      * Is There NextBackup For Time
901 1
902 1      * Function Description:
903 1      * Determine if a backup exists after to the specified time
904 1
905 1      * Parameters:
906 1      * context (I) - Pointer to the restore context
907 1      * thisTime(I) - Time for the query
908 1      * flags (I) - Backup constraint flags: e.g., full-only/partial-ok
909 1      * isThere (O) - TRUE/FALSE that requested backup does exist
910 1
911 1      * Return Codes:
912 1      * E_SUCCESS - operation completed successfully
913 1      * EP_RE_RECOVER_xxx - when errors occur accessing catalogs
914 1
915 1      *
916 1
917 1      restore_ly
918 1      RSTPL_IsThereNextBackupForTime( restore_context
919 1      *context,
920 1      *context,
921 1      RSTPL_Backup_flags_ly
922 1      flags,
923 1      boolean_ly
924 1      );
925 1
926 1      #ifdef _cplusplus
927 1      }
928 1      #endif
929 1
930 1      #endif /* #ifndef H_RSTPLAPI */

```